



**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

---

**Факултет по телекомуникации**

**Катедра „Радиокомуникации и видеотехнологии“**

**Специалност: телекомуникации**

# **ДИПЛОМНА РАБОТА**

**Тема:** Проектиране и изграждане на система за контрол на достъп чрез RFID.

Студент: Васил Валентинов Орешенски

Фак. No: 111209158

Научен ръководител: Доц. д-р Иво Дочев

София, 2020

# Съдържание

ДИПЛОМНА РАБОТА.....	0
Списък фигури.....	2
Списък таблици .....	3
Увод .....	4
Глава 1. Състояние на проблема по литературни данни.....	5
1.1. Представяне на системите за контрол на достъп.....	5
1.2. Защо Arduino базиран четец и Wi-Fi комуникация ? .....	11
Глава 2. Теоретична част .....	13
2.1. Модул за осигуряването на контрол на достъп .....	13
2.2. Модул за Администрация .....	23
2.3. Комуникационна сигурност.....	27
2.4. Лог (Log) .....	30
3. Инженерно решение на поставената задача .....	31
3.1. Четец.....	31
3.2. REST API.....	42
3.3. Website.....	46
3.3. База данни .....	54
4. Анализ на получените резултати, приложимост и изводи .....	58
4.1. Бързодействие на системата .....	58
4.2. Сигурност.....	59
4.3. Капацитет на системата.....	60
4.4. Гъвкавост при интегриране.....	61
Заклучение .....	62
Литература и интернет източници .....	63
Използвани съкращения .....	65
Изходен код и използван software .....	66

## Списък фигури

1. Фигура 1.1. Топология на СКД с централен сървър.....	11
2. Фигура 2.1. Блокова схема на четец, сървър и базов принцип на работа	14
3. Фигура 2.2. Схема на организацията в RFID антена.....	16
4. Фигура 2.3. Индуктивно-резонансен пренос на енергия .....	17
5. Фигура 2.4. URI стандарта разбит на основни компоненти.....	20
6. Фигура 2.5. Блокова схема на комуникация между browser и сървър	23
7. Фигура 2.6. Блокова схема на принцип на middle-man-attack.....	26
8. Фигура 3.1. Снимка на Arduino Uno REV3.....	30
9. Фигура 3.2. Снимка на MRFC522 RFID четец.....	31
10. Фигура 3.3. Снимка на MIFARE 1K Classic ключодържател .....	32
11. Фигура 3.4. Снимка на ESP-01 ESP8266 адаптер.....	33
12. Фигура 3.5. Снимка на LED.....	34
13. Фигура 3.6. Схема на свързване на четеца.....	38
14. Фигура 3.7. 2D чертеж на кутията от пред (вляво) и отгоре (в дясно)	39
15. Фигура 3.8. 2D чертеж на капака на кутията гледан от пред.....	39
16. Фигура 3.9. 2D чертеж на капака на кутията гледан от горе.....	40
17. Фигура 3.10. 3D модел на кутията.....	41
18. Фигура 3.11. Кутия принтирана от 3D принтер.....	41
19. Фигура 4.1. Снимка на пакетите между четеца и сървъра анализирани чрез WireShark.....	58
20. Фигура 4.2. Снимка на пакетите между browser и сървъра анализирани чрез WireShark.....	59

## Списък таблици

1. Таблица 2.1. Честотни диапазони на RFID системите.....	15
2. Таблица 2.2. Различни версии на Wi-Fi.....	19
3. Таблица 3.1. Технически параметри на Arduino Uno REV3 .....	30
4. Таблица 3.2. Технически параметри на MRFC522.....	31
5. Таблица 3.3. Технически параметри на ESP8266.....	32
6. Таблица 3.4. Технически параметри на ESP-01 адаптер.....	33
7. Таблица 3.5. Технически параметри на LED .....	34
8. Таблица 3.6. Разположение на щифтове при MRFC522 .....	34
9. Таблица 3.7. Разположение на щифтовете при ESP8266.....	35

## Увод

Системите за контрол са неделима част от инфраструктурата на почти всяка бизнес сграда в днешно време. Те се делят на няколко категории според вида на данните нужни за осъществяване на контрола на достъп и според инфраструктурната им реализация. С напредването на компютърните и мрежови технологии се появяват и така наречените IP базирани системи. Отношението към тях остава консервативно основно заради по – сложната начална конфигурация и по – високата цена.

Целта на настоящата дипломната работа е реализация на централизирана безжична система за контрол на достъпа чрез RFID базиран четец, сървърна част за администриране и осъществяване на идентификационен процес чрез Wi-Fi комуникация между четеца и сървъра.

За да бъде постигната целта на настоящата дипломна работа са поставени следните задачи:

1. Изграждане на RFID четец на базата на Ардуино борд (англ.: Arduino board) за разчитане на данни от пасивен RFID идентификатор
2. Моделиране на кутия за RFID четеца
3. Разработване на програмен продукт състоящ се от 2 модула
  - 3.1. Модул за администрация със следните възможности
    - Регистриране на идентификатор
    - Премахване на идентификатор
    - Деактивиране на идентификатор
    - Регистриране на точна за достъп
    - Премахване на точка за достъп
    - Деактивиране на точка за достъп
    - Генериране на справки
    - Експорт на данни
  - 3.2. Модул за Wi-Fi комуникация м/у четеца и сървъра за осъществяване на контрола за достъп.

# Глава 1. Състояние на проблема по литературни данни

## 1.1. Представяне на системите за контрол на достъп

Системите за контрол на достъп (СКД) имат за цел да ограничат достъпа до даден ресурс. В зависимост от ресурса може да говорим за ограничение на достъп до софтуерни данни или ограничаване на физическия достъп до даден ресурс. Тъй като двете системи се различават коренно в своите принципи на работа текущата дипломната работа има за цел да изгради СКД до физически ресурс. В най – общата си форма една СКД се състои от следните компоненти: четец, идентификатор, механизъм за ограничаване на достъпа, контролен панел и система взимаща решения за контрола на достъп.

### Идентификатори

Идентификатора представлява физически обект, парола, биометрични данни на дадено лице или даден вид информация нужна за да се даде достъп до ресурс. Според вида и начина на доставяне на информацията това може да са както следва и не само:

- **Биометрични:** пръстов отпечатък, отпечатък на ръка, лицево разпознаване, разпознаване на ириса на окото, сканиране на ретината на окото, гласово разпознаване.
- **Информация:** пин код, комбинация от числа и букви (парола).
- **Контактни карти:** Карти с магнитна лента, Smart карти.
- **Безконтактни карти:** RFID / NFC базирани карти.
- **Други:** баркод.

## Четци

Четеца представлява входно устройство за данни, което има за цел прочитане на данните от идентификатор и предаването им на системата за взимане на решение за контрола на достъпа. Вида на четеца се разделя на две основни групи - в зависимост от вида на данните, които се четат и според функционалностите на четеца.

### Четци според вида данни

**Баркод:** комбинация от редуващи се светли и тъмни райета (1D - едномерни), които се обработват от оптичен четец. Информация, която носят е на основата на морзов код, но вместо точка и запетая имаме тесни и широки райета. QR кодовете (2D - двумерни) представляват разновидност на баркодовете при които информацията е кодирана под формата на тъмни квадрати на бял фон, като информацията е кодирана както по вертикала, така и по хоризонтала.

- *предимства:* лесен и евтин начин за генериране на идентифициращ елемент, както и лесното му интегриране върху различен набор от предмети.
- *недостатъци:* ниска сигурност, лесен начин за дублиране на реални баркодове, както и генериране на фалшиви.

**Биометрични:** сканиране на физически атрибут на дадено лице. Сканирането генерира модел, който бива сравнен с образец. Има два режима на работа – 1-към-1 (1-to-1) или 1-към-много (1-to-many). При 1-към-1 освен сканиране на физически атрибут, потребителя е длъжен да въведе парола или да се идентифицира с карта. Сканирания модел бива сравнен с образец асоцииран с паролата – по този начин се постига голямо бързодействие, тъй като модел бива сравнен само с един образец, постига се и по – висока сигурност заради

допълнителния начин за идентификация. При 1-към-много сканирания модел се сравнява с всички образци (често равен на броя на потребителите).

- *предимства*: висока сигурност, елиминиране на възможност за изгубена карта, забравена парола, карта дадена на заем
- *недостатъци*: при 1-към-много режим на работа имаме сравнително бавна обработка – сравнение на до 3000 модел за секунда.

**Информационни:** четеца е снабден с клавиатура, която се използва за въвеждане на нужната информация (най – често парола).

- *предимства*: евтина цена за разработка, лесно програмируема
- *недостатъци*: забравена парола, ниска сигурност

**Кarti с магнитна лента:** картата има магнитна лента изградена от метални частици. Информацията се записва чрез промяна на магнитите свойства на тези частици, като магнитната лента се състои от три пътеки и всяка може да бъде кодирана по различен начин. В четеца има бобина (solenoid) и при физически контакт между картата и четеца се индуцира напрежение.

- *предимства*: евтина цена за изработка, лесно програмируема
- *недостатъци*: податливи на износване, грешно прочитане на информация, широкото им разпространение и стандартизираните подходи на кодиране на информация е довело да изграждането на устройства, които се поставят върху оригиналния четец с цел прочитане на данните.

**Wiegand:** Името идва от така наречения Wiegand ефект (на името на откривателя John R. Wiegand). Ефекта се наблюдава при жичка направена от феромагнитна сплав между кобалт, желязо и ванадий. Самата жичка е направена от по – твърда обвивка с висока корцетивност и ,мека‘ сърцевина.



Ефекта се изразява в това, че при прилагане на магнитно поле външната обвивка предпазва сърцевината от намагнитване до достигна на магнитния праг (magnetic threshold) на външната обвивка – при което цялата жичка (сърцевина и външна обвивка) променят своята полярност на магнетизацията.

- *предимства:* тъй като жичките са вградени в картата тук няма износване, трудно дублиране - висока сигурност
- *недостатъци:* изместени от безконтактните четци заради по – добрата устойчивост на външна намеса и удобството на безконтактната карта

**Безконтактни** – четеща и картата комуникират чрез радио вълни на определена честота в зависимост от нуждите (скорост и обем на четене на данните, разстояние между идентификатора и четеща). Картите се разделят на два основни вида според начина на комуникация – активни и пасивни. Четеща и картата комуникират чрез принцип наречен индуктивно-резонансен пренос на енергия. Пасивните карти имат три компонента, които са вградени в самата пластика на картата – антена направена под формата на бобина, кондензатор и интегрална схема, която съдържа идентификационната информация в определен формат. Четеща също има антена, която непрекъснато излъчва радио вълни. Когато картата е поставена в обхвата на четеща бобината и кондензатора на картата поглъщат и съхраняват енергията от полето. Тази енергия се обръща до постоянен ток, който захранва интегралната схема. Интегралната схема изпраща информацията, която е съхранена на нея до антената, която изпраща от своя страна информацията то четеща посредством радио вълни. При активните имаме литиева батерия служеща като източник на енергия. Интегралната схема съдържа приемник, който използва батерията за да усили сигнала, също така предавателя

използва батерията за да усилва сигнала от карата, което позволява комуникация от по – голямо разстояние.

- *предимства:* комуникация от разстояние, възможност за пренос на голям обем данни, протоколи за работа с няколко карти наведнъж.
- *недостатъци:* възможност за не-оторизиран достъп до данните на картата от разстояние
- 

### **Четци според функциите, които изпълнява**

**Основни (не-интелигентни):** разчитат идентификатора и предават информацията към контролния панел, не се грижат взимането на решение за контрола на достъп, също така, не отговарят за осъществяването на контрола. Най – често се използва Wiegand протокола за пренос на данните до контролния панел, но има и алтернативи като RS-232, RS-485. Това са най – разпространените видове четци.

**Полу-интелигентни:** освен разчитането на идентификатора, те разполагат и с всички механизми нужни за осъществяването на контрола на достъп, но не взимат решение за самия достъп. След прочитане на идентификатора информацията се изпраща до контролен панел, който взема решението за контрола на достъп. За комуникацията между четеца и контролния панел най – често се използва RS-485. При проблема комуникация между двете устройства, четеца спира да работи.

**Интелигентни:** освен разчитането на идентификатора, те разполагат с всички механизми нужни за осъществяването на контрола на достъп както полу-интелигентните четци, освен това притежават памет и процесор, което им позволява да взимат решения относно контрола на достъп. Свързани са към контролен панел посредством RS-485. Ролята на контролния панел е да

синхронизира периодично информация за потребителните от сървъра с тази в четците. По – този начин при проблем при комуникацията между четца и контролния панел, четца ще продължи да работи.

- **IP:** IP четците са вид интелигентни четци, като основната разлика при тях е липсата на контролен панел. Комуникират директно със сървъра на базата на Ethernet или Wi-Fi.

### **Механизъм за ограничаване на достъп**

Механизъм за ограничаване на достъпа. Когато говорим за контрол на достъп до помещение това би бил механизъм за отключване и заключване на врата, който бива управляван директно от четца или от контролен панел.

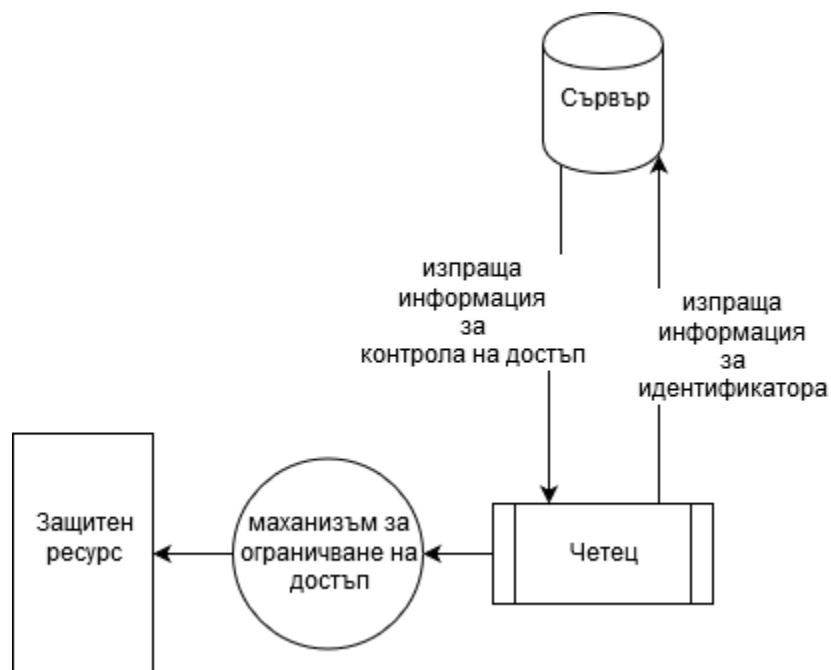
### **Система взимаща решения за контрола на достъп**

Устройство, което служи за съхранение и манипулиране на база данни или дава възможност за комуникиране с такава база данни и възможност за сравнение на информация за идентификация спрямо тази в базата данни и предаването на резултат от сравнението посредством даден протокол за комуникация.

### **Топология**

След прочитането на идентификатора, четца изпраща информацията за сравнение с база данни. Сравнението може да бъде направено от сървър, контролен панел или от самия четец. Информацията от сравнението се връща обратно до устройство, което контролира достъпа било то контролен панел или четец. В зависимост от резултата от сравнението потребителя получава съответното ниво на достъп или му бива отказано такова.

Обмяната на информация се осъществява чрез дадена среда за разпространение, съответния протокол за комуникация и стандарт за кодиране на данните.



Фигура 1.1. Базова топология на система за контрол на достъп с централен сървър.

## 1.2. Защо Arduino базиран четец и Wi-Fi комуникация ?

В днешно време най – разпространени решения са напълно автономните четци, които съдържат в себе си база данни и сами взимат решението за контрол на достъп. Най – често работят с парола или RFID чип. Те се ползват предимно в сгради с единствен вход. Един от основните недостатъци на тези решения, е че не предлагат анализ на данните.

Другия вид широко използвано решение е четец свързан към IP контролен панел, който от своя страна комуникира със сървър. Това решение е по – подходящо за сгради с в повече от един вход, също така тези решения

предлагат възможност за анализ на данните като много често това е под формата на услуга, която се заплаща допълнително. Недостатък на тези решения е нуждата от LAN окабеляване между контролера и сървъра. Нуждата от контролен панел също може да бъде разгледана като недостатък заради допълнително оскъпяване и централизацията на потока на данни – ако контролера бъде повреден всички четци свързани към него спират да работят.

Предимствата на Ардуиното са голямата модулност, което позволява по – качествено преизползване на отделни компоненти например: лесно заменяне на Wi-Fi модула без да се налага промяна по модула за RFID четеца или пък смяната на RFID четеца с биометричен без да се налага смяна на Wi-Fi модула. Друго предимство е лесното управление и ниската цена.

Wi-Fi комуникацията решава проблема с LAN окабеляването както и възможността за преизползване на вече изградени такива мрежи.

## Глава 2. Теоретична част

Дипломната работа се състои от на два самостоятелни логически модула. Модул за осигуряване на достъп състоящ се от четец и сървърна част, и модул за администрация състоящ се от website обслужван от сървърна част.

### 2.1. Модул за осигуряването на контрол на достъп

#### Принцип на работа

При доближаване на чипа до четеца на максимално разстояние от няколко сантиметра, той бива захранен от електромагнити вълни на честота от 13.56 MHz изпращани от четеца благодарение на принципа за индуктивно-резонансен пренос на енергия [1], при което чипа изпраща обратно своята идентификационна информация чрез електромагнитна вълна. RFID модула декодира информацията, след което бива предадена на ардуиното, което от своя страна изпраща HTTP GET заявка, която съдържа идентификационната информация за чипа и уникален идентификационен номер за ардуиното, към REST API-то посредством Wi-Fi модула. Сървър и четец са свързани към една и съща Wi-Fi мрежа. Заявката се изпраща в JSON формат [2]

Сървърът сравнява получената информация със информацията налична за четец в база данни. В базата данни се пази информация за минималното ниво на достъп нужно на даден потребител за да му бъде позволен достъп. REST API-то и базата данни комуникират помежду си чрез SQL комуникационен протокол [3], който работи на базата на TCP/IP протокола.

В базата се пази информация за нивото на достъп на всеки потребител / идентификатор както и минималното ниво на достъп, което изисква дадена точка за достъп (четец) за да бъде допуснат потребителя през нея.

Сървърът връща информация от сравнението в текущата HTTP заявка, която зависи от това дали е намерена информация за четец и потребителя в базата

данни, и дали нивото на достъп на потребителя е достатъчно. Когато потребителя има нужното ниво на достъп се връща HTTP отговор с код 200 (OK), когато няма достъп се връща 401 (Unauthorized), а когато няма информация за потребителя се връща 404 (Not found).

Получения отговор от сървъра се декодира от Wi-Fi модула и се предава като информация на ардуиното, което от своя страна дава индикация дали потребителя има право да достъпи защитения ресурс.

При всяка заявка от четеца към сървъра в базата се прави запис за събитието, като в последствие тази информация се ползва от модула за администрация за анализ на данните под формата на генериране на справки.

За осигуряване на качествена поддръжка на класическа система работеща с база данни, мрежи се генерира подробен log файл при всяка възникнала грешка на сървъра.

За осигуряването на висока защита на комуникация между четеца и сървъра се използва разновидност на протокола HTTP – HTTPS. Защитата се осигурява чрез криптиране на данните в двете посоки.

## **Блокова схема**



Фигура 2.1. Блокова схема на комуникацията между Wi-Fi базиран ардуино четец и REST API посредством HTTP протокол

## RFID

RFID е безжична система за идентификация чрез радио вълни. Състои от четец и идентификатор. В зависимост от нуждите на системата RFID се дели на няколко честотни диапазона.



Таблица 2.1. Честотни диапазони на RFID системите

<b>Честотен диапазон</b>	<b>Разстояние</b>	<b>Скорост на трансфер</b>	<b>Приложение</b>
120 – 150 kHz ниско честотни (LF)	До 10 см	ниска	Идентификация на животни и предмети
13.56 MHz високо честотни (HF)	До 1 м	ниска до средна	Идентификация, контрол на достъп, инвентаризация, обмен на данни
433 MHz утра-високо честотни (UHF)	1 – 100 м	средна	Контрол на достъп в комбинация с активен таг
865 – 868 MHz (Европа) 902 – 928 MHz (САЩ) ултра-високо честотни	1 – 12 м	средна до висока	Използва се при релсови пътища
2450 – 5800 MHz микровълнови	1 – 2 м	висока	802.11 WLAN Bluetooth
3.1 – 10 GHz микровълнови	До 200 м	висока	

Според начина на захранване на идентификатора RFID се дели на активна и пасивна система. Текущата дипломна работа има за цел изграждането на пасивна система.

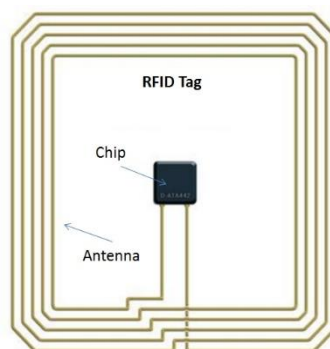
При пасивната система идентификатора се захранва чрез пренос на енергия от четеща. Използват се за комуникация от близко разстояние.

**Идентификатор** - в най – простия си вид се състои от три елемента:

интегрална схема, която се използва за съхранение, обработка на информация и модулация и демодулация на радио сигнала. Начин за преобразуване на сигнала от четеца до постоянен ток и антена за получаване и предаване на сигнал. Информацията на идентификатора се съхранява на енерго-независима памет. Елементите се вграждат най – често в самата пластика на карата, или в пластмасови обекти пригодени за прикачване към ключодържател.

Пасивните идентификатори нямат батерия и тяхното захранване зависи изцяло от силата на сигнала от четеца. Информацията която се съдържа най – често е сериен номер. Сериения номер се записва по време на производствения процес, като той няма връзка с конкретен обект или лице, а в последствие тази информация се установява най – често в база данни, към която се записват обработените данни. Има и пасивни идентификатори, които подлежат на манипулация и тяхната информация може да бъде презаписана. Скоростта на четене от тези чипове е относително ниска, като подобна операция отнема между 25 до 50 ms.

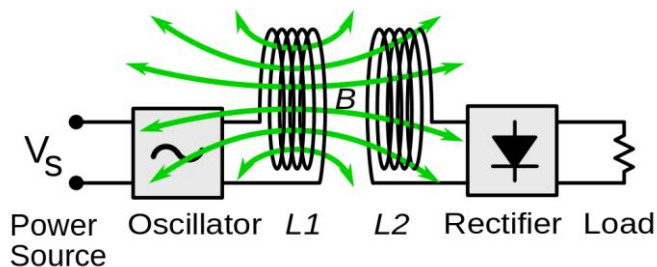
Антената на идентификатора събира енергия и я пренасочва към интегралната схема, като по този начин я захранва. Обема на енергията пряко зависи от размера и формата на антената. Също така антената е отговорна и за предаването на информацията, която е запазена върху идентификатора.



Фигура 2.2. Схема на организацията в RFID антена

**Четец** – той е отговорен за захранването на идентификатора, разчитането на данните изпратени от него и предаването им към сървърната част.

Начина на комуникация и захранване става на принципа на резонансно-индуктивен пренос на енергия. Антената на четеца се състои от бобина свързана към източник на променлив ток като по този начин се образува магнитно поле около него. Когато поставим идентификатора в близост до полето се създава електро-моторна сила по закона на Фарадей за електромагнитна индукция. Напрежението води създаване на електрически поток в бобината на идентификатора. Посредством токоизправител променливия ток се преобразува до постоянен, чрез който се захранва интегралната схема. При което антената на идентификатора също създава магнитно поле, което бива разчетено от антената на четеца. Двете антени са настроени да резонират към една и съща резонансна честота, което позволява по – оптимален пренос на енергия на по – големи разстояния.



Фигура 2.3. Индуктивно-резонансен пренос на енергия

## Wi-Fi

Представява технология за безжична комуникация базирана на IEEE 802.11 стандарта [4], който се използва за локална мрежова комуникация (LAN) и за достъп до интернет. [5]. Wi-Fi комуникацията използва пакети за пренос на данни, които се предават чрез радио вълни между отделните устройства. Това се постига чрез модулация и демодулация на носещата вълна. По стандарт

всяко устройство разполага с MAC адрес, който представлява 48 битов уникален адрес. MAC адреса се използва за обозначаването на източника и получателя за даден пакет.

Самата технология следва препоръките на OSI модела, но е направена с идеята да е съвместима с Ethernet стандарта, който се използва за пренос на TCP/IP [6] трафик, който от своя страна не следва OSI модела.

TCP/IP [6] модела е създаден преди дефинирането на OSI модела. Той дефинира стандарти за пакетиране, адресиране, пренос, пренасочване и получаване на данни между клиент-сървър в една компютърна мрежа и е един от най – разпространените в днешно време. Функционалността на тези протоколи е организирана в четири абстрактни слоя, към които са класифицирани съответните протоколи според тяхната роля.

Двата най – важни протокола са съответно Transmission Control Protocol (TCP) и Internet Protocol (IP).

TCP - дефинира надежден начин за пренос на данни под формата на пакети между приложения. Едни от най – използваните приложения в днешно време като World Wide Web (WWW) и емайл клиентите използват този протокол.

Протокола гарантира получаване на изпратените пакети такива каквито са били изпратени от сървъра и тяхното им получаване в правилен ред за да може да се възстанови изпратеното съобщение. При възникване на грешка в мрежата или претоварване, някои пакети могат да бъдат изгубени или получени в грешен ред при клиента. Протокола засича тези проблеми и използва техника позната като ,позитивно потвърждение‘ (positive acknowledgment) с повторно изпращане на данните. Това задължава получателя на данните да изпрати потвърждение, че пакетите са получени

успешно. Ако проблема остане не разрешен, първоизточника на данни бива известен за проблема.

Таблица 2.2. Различни версии на Wi-Fi.

Стандарт	Максимална скорост	Година на експлоатация	Честота
Wi-Fi 6 (802.11ax)	600 - 9608 Mbit/s	2019	2.4 / 5 GHz 1 - 6 GHz ISM
Wi-Fi 5 (802.11ac)	433 - 6933 Mbit/s	2014	5 GHz
Wi-Fi 4 (802.11n)	72 - 600 Mbit/s	2009	2.4 / 5 GHz
802.11g	3 – 6 Mbit/s	2003	2.4 GHz
802.11a	1.5 – 54 Mbit/s	1999	5 GHz
802.11b	1 – 11 Mbit/s	1999	2.4 GHz

Wi-Fi е маркетинговото име на устройствата, които имплементират горе посочените стандарти.

## HTTP

Протокол за пренос на данни между сървър и клиент. Работи на Application слоя както на OSI модел така и на TCP/IP модела. HTTP е основния протокол, който се използва от WWW. Основните данните, които се пренасят са формати свързани с изграждането на Web страници: HTML документи, JavaScript файлове, CSS файлове, JSON документи, картинки, видеа, музика и други медия файлове [7], както и информация предназначена за сървъра от клиента: най – често под формата на JSON документи.

За да бъде достъпен един ресурс, той трябва да бъде идентифициран. За тази цел всеки ресурс е представен чрез уникален ресурсен идентификатор –

Uniform Resource Identifier (URI). URI формат е текстово базиран и включва в себе си няколко основни елемента.

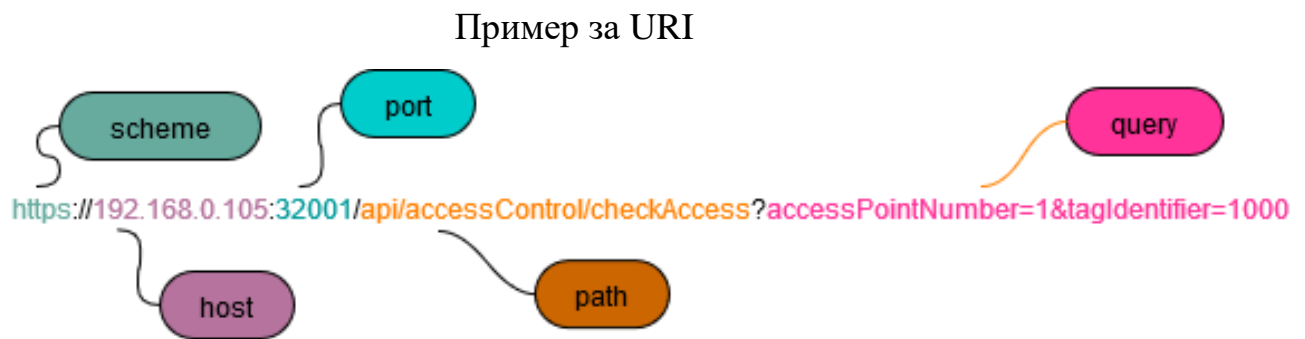
**Scheme:** вида на протокола който се използва

**Host:** IP адрес или домейна

**Port:** порт на който сървърът обслужва дадения website или REST API

**Path:** фрагменти разделени с дясно наклонена черта '/'. Обикновено са описателни текстове, който дават информация за същността на ресурса

**Query:** комбинация от двойка ключ и стойност допълващи информацията за целта на заявката.



Фигура 2.4. URI стандарта разбит на основни компоненти

Клиента и сървърът комуникират чрез размяна на съобщения. Съобщенията изпратени от клиента се наричат request съобщения, а тези изпратени от сървърът се наричат response съобщения. Протокола е текстово базиран с цел по – лесното разчитане от човек.

Всяко request съобщение дефинира информация относно целта на заявката под формата на полета (headers) [8] състоящи се от име и стойност, и тяло на заявката в зависимост от вида на HTTP метода. Където най – често срещаните са: accept, accept-encoding, cookie, host, user-agent, content-length, content-type и други.

За да се окаже какво действие искаме да извършим на дадения ресурс, HTTP протокола дефинира така наречените методи (HTTP methods / verbs):

**GET** – заявката очаква даден ресурс да бъде върнат от сървъра в определен вид и нищо повече.

**HEAD** – заявката е същата като **GET**, но без да се връща самия ресурс.

Полезно е когато искате информация за мета данни за ресурс, но без да ви трябва самия ресурс. Например: може да се върне информация колко е голям един файл, без да се налага целия файл да бъде част от response съобщението.

**POST** – заявката се състои от тяло и оказва на сървъра, че тялото на съобщението трябва да бъде третиран като нов ресурс, т.е. след успешна заявка, новия ресурс ще има собствен URI. Например: най – често се ползва, когато искате да запазите нов запис в базата данни на сървъра.

**PUT** – също като **POST**, оказва на сървъра, че тялото на заявката трябва да създаде нов ресурс под съответното URI. Различава се с това, че ако ресурс вече съществува, се очаква сървъра да модифицира стойността за това URI с тази изпратена от заявката.

**DELETE** – оказва на сървъра, че ресурса със съответното URI трябва да бъде изтрит.

**PATCH** – оказва на сървъра, че искаме изцяло или частично да променим ресурса на определено URI със ресурса, който изпращаме като част от заявката.

Response съобщенията също дефинират набор от полета (headers) [8]: cache-control, content-encoding, content-length, content-type и други.

За да се окаже какъв е резултата от response заявката, в request-a HTTP протокола дефинира Status code. Те са числови стойности, които се разделят на пет групи в зависимост от числото с което започва:

**1XX** – информационни.

**2XX** – оказват, че заявката е била успешна. Пример: 200 – ОК

**3XX** – оказват, че заявката е пренасочена. Пример: 301 – оказва, че търсения ресурс е вече се намира на друго URI.

**4XX** – оказват, че заявката е невалидна спрямо очакванията на сървъра.

Пример: 404 – оказва, че поискания ресурс вече не съществува

**5XX** – оказват, че на сървъра е възникнала грешка. Пример: 500 – грешка на сървъра.

## **JSON**

Формат за описване на данни. Силно заложен в HTTP комуникацията и е един от основните формати за обмен на данни между клиент и сървър.

Представява текстов формат имащ за цел да бъде лесно разчитан както от човек така и от компютър, като едновременно с това се опитва да е с минимален размер. Това е постигнато чрез просто представяне на данните под формата на колекция от ключ и стойност, където ключа е текст а стойност може да е текст, списък или друг JSON документ.

Пример за JSON обект, който описва информация за потребител:

```
{ "firstName": "vasil", "lastName": "oreshenski", "age": "29" }
```

## **2.2. Модул за Администрация**

### **Принцип на работа**

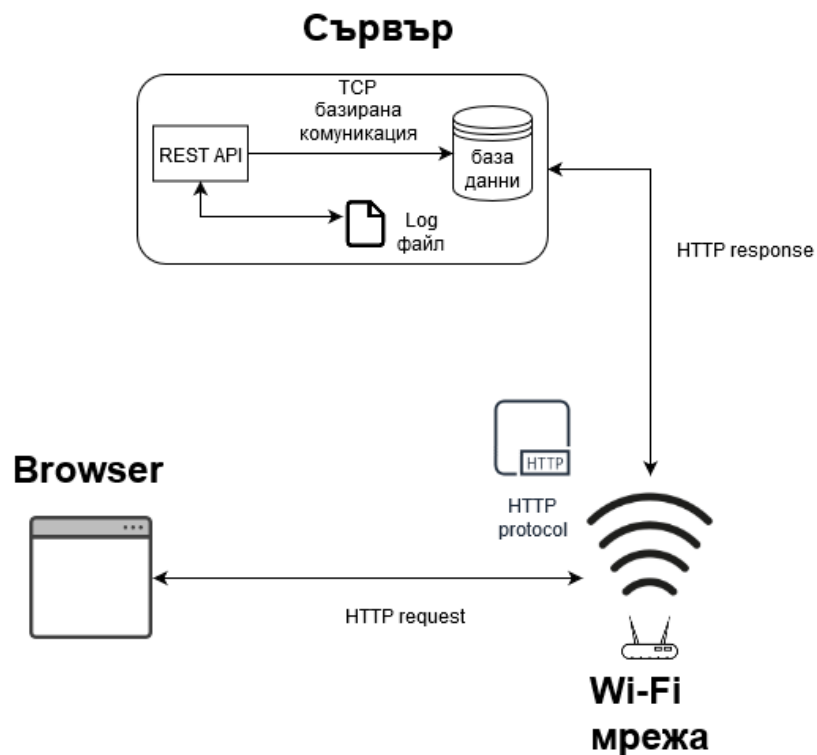
Модула е реализиран под формата на website обслужван от сървър на определен адрес и порт. За да може потребителя да достъпи website-a, те трябва да се в една мрежа. Website-a дава възможност за администриране на идентификаторите и точките на достъп. Под точка на достъп се разбира четец, но от потребителска гледна точка този термин е по - подходящ.



Комуникацията се извършва на базата на HTTPS протокола.

При първоначално отваряне на website-a, потребителя ще бъде пренасочен към страница за автентикация, където ще трябва да въведе потребителско име и парола, след успешна автентикация, потребителя е пренасочен към началната страница на website-a.

### Блокова схема



Фигура 2.5. Блокова схема на комуникация между browser и сървър

### Автентикация

За да бъде предпазен website-a от не желана външна намеса са взети няколко мерки. Пренасочване на не-автентикирани потребители към страницата за въвеждане на потребителско име и парола, и запазване на нужната информация за автентикиране при последващи заявки към сървъра при клиента. Това е постигнато чрез JSON Web Token (JWT) [9] стандарта.

JWT предоставя лесен и компактен подход за автентикация, чрез обмен на съобщения под формата на JSON. Информацията, която се пренася винаги се подписва с таен ключ, който е познат само на сървъра генерирал token-а. Чрез подписване сървъра може да провери при следваща заявка дали token-а бил подписан със същия ключ и съответно да предприеме нужното действие ако не е. Това ни предпазва от възможността, външен потребител да изфабрикува фалшив token, тъй като той не знае ключа с който трябва да бъде подписан за да се валидира от сървъра. Подписването се извърша чрез HMAC алгоритъм [10].

Стандарта се състои от три части: Header, Payload и Signature.

**Header** – обикновено съдържа информация за типа на token-а и вида на алгоритъма използва при подписването му. Например:

```
{ "alg": "HS256", "typ": "JWT" }
```

**Payload** – съдържа информация свързана с потребителя като уникален номер в базата, роля (например: администратор). Например:

```
{ "id": "126", "role": "admin" }
```

**Signature** – Header-а и Payload-а се кодират чрез Base64 [11] алгоритъм и се криптират с посочения алгоритъм в Header-а и тайния ключ и получения резултат представлява Signature-а.

Накрая се получава текстов формат, който представлява самия token.

Например:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmlxdWVfbmFtZSI6InRlc3RAdGVzdC5jb20iLCJyb2xlIjoiaWoiQWRtaW4iLCJuYmYiOiJlNzgzNDcwMzQsImV4cCI6MTU3ODg0NzIxNCwiaWF0IjoxNTc4ODQ3MDM0fQ.zk3wJXnPm6RL-yQ17xJ7vHNwI0wvtxBD7hQBXE0e-JE
```

Интегрирането на token-а в HTTP протокола става чрез authorization header.

След като потребителя въведе валидни потребителско име и парола в response

съобщението се връща и token, който се запазва с browser-а докато не бъде затворен. В header-те на всяка следваща заявка към сървъра се включва token-а, по този начин на потребителя няма да му се налага да въвежда потребителско име и парола след всяко действие.

С цел допълнителна сигурност token-а се генерира с определена продължителност на живота. След изтичането на този период, token-а става невалиден и потребителя бива пренасочен към страницата за въвеждане на потребител и парола, при което му се генерира нов token.

### **Потребителска парола**

При работа с потребители, е важно паролата да не бъде запазвана в чист вид в базата данни от гледна точка на сигурност. При регистриране върху паролата на потребителя се прилага подход наречен hashing.

Hashing има за цел да видоизмени стойността по такъв начин, че да е невъзможно да бъде възстановена оригиналната стойност. Един от най – популярните алгоритми за hashing е SHA (Secure Hash Algorithms).

По – този начин ако базата данни или данните в нея попаднат в грешни ръце, чувствителната информация като пароли ще бъде неизползваема.

Тъй като във времето е имало не малко на брой откраднати данни от не защитени сайтове също така има и публично достъпни данни за hash-те стойности на най – често използваните пароли, е възможно злонамерен потребител да опита да налучка паролата на даден потребител чрез атака наречена pre-computed hash attack (атака с предварително генериран hash).

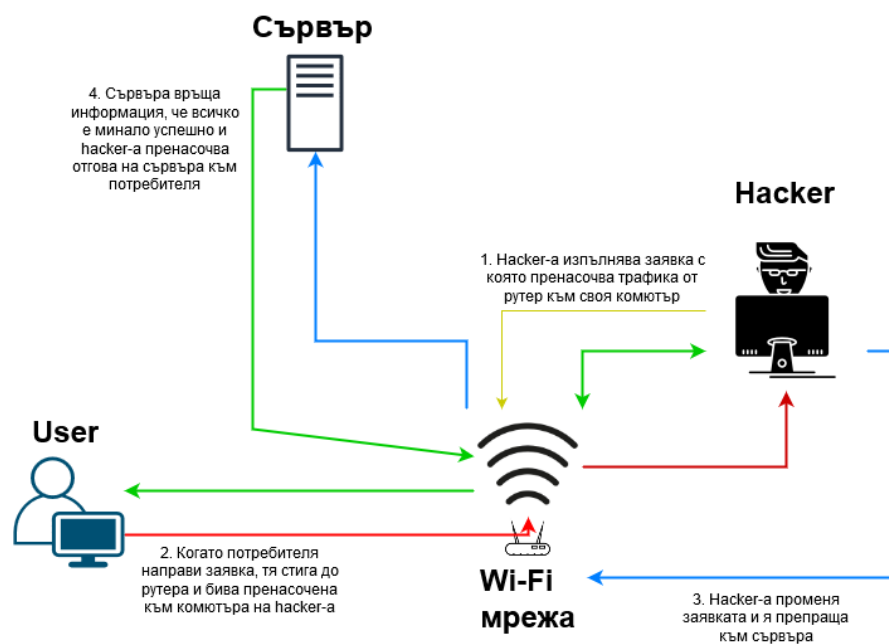
Това е сериозен проблем за сигурността, защото потребителите често използват една и съща парола за всичките си регистрации и е достатъчно да се регистрирате в един не защитен сайт, при което всичките ви останали регистрации също биват застрашени от злоупотреба.

За да се избегне това в днешно време hashing алгоритмите използват допълнителна произволна стойност (salt) при генерирането на hash-a, която се добавя към паролата на потребителя.

### 2.3. Комуникационна сигурност

За да се постигне цялостна сигурност на комуникация се използва HTTPS комуникация между четеца и сървър, както и между web browser-a и сървъра (в контекста на модула за администрация).

Един от недостатъците на Wi-Fi мрежите, е че са много по – лесно достъпни от колкото LAN мрежата, където е нужна физическа връзка чрез кабел. Това отваря възможност за така наречената middle-man-attack. Тя се изразява във възможността на даден потребител да подслушва или манипулира трафика, който се предава през рутер или друго комуникационно устройство, чрез пренасочване на този трафик до сървър / компютър, до което той има пълен контрол. Това става проблем с масовото навлизането на Wi-Fi мрежите в последното десетилетие.



Фигура 2.6. Блокова схема на принцип на middle-man-attack

Частично проблема може да се реши като се защити създадената мрежа от рутера чрез парола, което би попречило на външен потребител, който не знае паролата, но проблема остава наличен за потребители, които вече знаят паролата или когато мрежата към която са свързани е публично достъпна. Една от основните задачи на HTTPS е да реши този проблем.

HTTPS има за цел да осигури защитена комуникация между клиент-сървър, чрез криптиране на трафика. HTTPS стъпва върху Transport Layer Security (TLS). За да се установи TLS връзка, клиента и сървъра изпращат редица съобщения с които се договарят за това какъв алгоритъм за криптиране ще се използва и се установява автентичността на сертификата представен от сървъра.

Първоначално клиента изпраща заявка до сървъра, в която оказва, че иска да използва TLS комуникация. Във заявка клиента е оказал различните версии на протокола, hash функции и видове алгоритми за криптиране, които поддържа. Обикновено това пряко зависи от версията на операционната система и версията на клиентското приложение, което се използва за комуникация (пример: Firefox v60.0.1). След като съобщението достигне до сървъра, според неговите възможности, избира възможно най – сигурните версия на протокол, hashing и криптиращ алгоритъм, като това отново до голяма степен зависи от версията на операционна система на сървъра и сертификата, който е приложен към него и с ново съобщение оказва на клиента какви алгоритми ще се използват за комуникацията.

След това сървъра се идентифицира пред клиента като изпраща асиметричен публичен цифров сертификат (digital certificate) предназначен за криптиране на web трафик.

Асиметричният сертификат представлява файл, който съдържа името на сървъра, до кога е валиден, сертифициращ орган (certificate authority),

публичен ключ (public key), таен ключ (private key) и какви алгоритми поддържа. При асиметричните алгоритми за криптиране, публичния ключ се ползва за криптиране, а тайния ключ се ползва за декриптиране. Този вид алгоритми се води еднопосочен, тъй като тайния ключ е известен само на притежателя на сертификата, а на всички останали се предоставя така наречения публичен сертификат (public certificate), който не съдържа в себе си тайния ключ. По този начин всички, които имат публичния сертификат могат да криптират данните и да ги изпращат до сървър, но само сървър има нужния ключ за да декриптира съобщенията.

Клиента проверява достоверността на публичния сертификат чрез така наречения сертифициращ орган (certificate authority). Всяка операционна система има предварително инсталирани сертификати наречени корен на сертификата (certificate root). За да се довери клиента на този публичен сертификат трябва операционната система да има инсталиран certificate root за конкретния сертифициращ орган. Едни от най – разпространените сертифициращи органи са VeriSign и DigiCert. Операционните системи (Windows, Linux, Mac OS) имат инсталирани certificate root от тези компании и когато сървър върне публичен сертификат със сертифициращ орган някоя от тези компании, той автоматично му се доверява, като това се случва на ниво операционна система или софтуерно приложение.

Следа размяна на ключове, които ще се използват за криптирането на трафика. Клиента генерира ключ под формата на произволен низ от символи (нарича се session key). Ключа бива криптиран с публичния ключ на сертификата и го изпраща до сървър. На този етап единствено клиента и сървър знаят не-криптираната стойност на този низ, тъй като клиента е този който го е генерирал, а след криптирането му и изпращането му до сървър само сървър знае как да го декриптира. Дори и да има потребител, който да

следи трафика, той не може да декриптира ключа тъй като само сървъра разполага с тайния ключ (private key) за декриптиране.

От този момент всяко съобщение изпратено от клиента или сървъра се криптира с ключа генериран от клиента (session key) в рамките на един връзка докато е отворена. При затваряне на връзката процеса се изпълнява на ново.

За да окаже клиента на сървъра, че иска да използва защитена връзка, обикновено това става чрез промяна на порта към който клиента иска да се свърже, като за HTTPS това е 443. По отношение интегрирането му в съвременните технологии, това става чрез промяна на URI схемата от http:// на https://.

В днешно време е честа практика, при опит за свързване със сървър по http, който е настроен да работи по https, той автоматично да пренасочи комуникацията с цел осигуряване на защита, тъй като това не коства нищо на потребителя и се случва без неговото пряко участие.

## **2.4. Лог (Log)**

Във всяка софтуерна система в различните и етапи на разработка, могат да се появят грешки по време на нейното функциониране. За да не се губи ценна информация всяка грешка се записва в така наречения log. В него се съдържа информация за грешката и състоянието на системата по време на нейното възникване, както и друга информация, която би показала причината заради, която е възникнала. Най – често под формата на файл.

## Глава 3. Инженерно решение на поставената задача

### 3.1. Четец

Четеца е изграден на базата на Arduino Uno REV3 модел [12], MFRC522 RFID модул [13], пасивен RFID идентификатор MIFARE Classic 1k [14] под формата на ключодържател и Wi-Fi модул ESP8266 [15]. Ардуиното играе ролята на управляващ елемент за Wi-Fi и RFID модулите. Двата модула са свързани към съответни цифрови щифтове на ардуино като комуникацията се осъществява чрез електрически сигнали.

#### Arduino Uno REV3



Фигура 3.1. Снимка на Arduino Uno REV3

Едноплатков микро-контролер базиран на ATmega328P. Може да бъде захранено чрез USB връзка или от външен източник чрез адаптер преобразуващ променлив към постоянен ток. Широко разпространен заради големия брой щифтове и разумна цена. Предварително зареден с bootloader, който позволява зареждане на нов програмен код без да се налага използването на програматор. Езиците за програмиране са C/C++. [12]

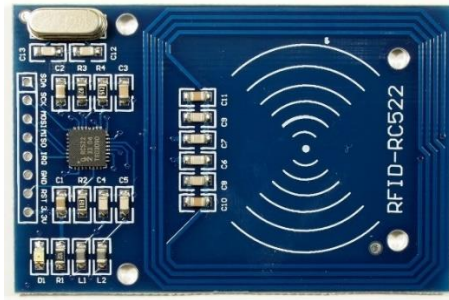
Таблица 3.1. Технически параметри на Arduino Uno REV3

Микро-контролер	ATmega328P
Работно напрежение	Работно напрежение 5V



Входно напрежение	6-20V
Цифрови щифтове	14 (от които 6 позволяват широчинно-импулсна модулация)
Щифтове с Широчинно-импулсна модулация	6
Аналогови щифтове	6
Постоянен ток на щифтовете	20 mA
Максимален толериран постоянен ток на входно-изходните щифтовете	40 mA
Постоянен ток на 3.3V щифт	50 mA
Флаш памет	32 KB (ATmega328P) от които 0.5 KB използвани от bootloader-a
Оперативна памет (SRAM)	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
CPU работна честота	16 MHz
Вградени LED индикатори	13
Дължина	68.6 mm
Широчина	53.4 mm
Тегло	25 g
Цена	20 €

## MFRC522 RFID



Фигура 3.2. Снимка на MRFC522 RFID четец

Интегрална схема предназначена за безконтактна комуникация на честота от 13.56 MHz. Поддържа стандартите ISO/IEC 14443 A/MIFARE и NTAG. [13]

Таблица 3.2. Технически параметри на MRFC522

Работен ток	13 – 26 mA
Работно напрежение	3.3 V
Работна честота	13.56 MHz
Разстояние за прочитане	0 ~ 35 mm
Дължина	60 mm
Широчина	40 mm
Цена	~ 5 €

## MIFARE Classic 1k идентификатор



Фигура 3.3. Снимка на MIFARE 1K Classic ключодържател

Семейство интегрални схеми за безконтактна комуникация по стандарт ISO/IEC 14443 A. Малките размери позволяват вграждането в пластмасови карти и ключодържатели. [14]

В текущата дипломна работа е използвана реализация под формата на ключодържател.

## ESP8266 Wi-Fi

Напълно завършен и самостоятелен Wi-Fi продукт с интегрирана поддръжка за TCP/IP протокола. Модула е способен самостоятелно да обслужва софтуерен продукт или да се използва като допълнение към друго устройство. Модула се разпространява с firmware, който поддържа набор от инструкции за управление наречени AT Commands. [15]

Таблица 3.3. Технически параметри на ESP8266

Поддържани стандарти	802.11 b/g/n, Wi-Fi Direct, Soft-AP
Работно напрежение	3.3 V – 3.6 V
AT Commands	Да, част от firmware
TCP/IP протокол	Да, интегриран
Цена	~ 5 €

## ESP-01 адаптер с регулатор на напрежение



Фигура 3.4. Снимка на ESP-01 ESP8266 адаптер

Адаптер за ESP8266 Wi-Fi модула използван като регулатор на входното напрежение до 3.3 V. [16]

Макар и ардуиното да предлага 3.3 V изходно напрежение, не винаги достига до тази стойност и тъй като ESP8266 е чувствителен на подобни падове, не винаги работи с пълни възможности. За да използва 5 V напрежение като

захранване е нужен регулатор на напрежение, тъй като толеранса на ESP8266 е до 3.6 V.

Таблица 3.4. Технически параметри на ESP-01 адаптер

Входно напрежение	5 V – 12 V
Изходно напрежение	3.3 V
Цена	~ 2 €

### Индикация за пропуск



Фигура 3.5. Снимка на LED

Използва се LED за да се окаже на потребителя дали достъпа е разрешен. Зелен диод за да се окаже успешен пропуск и червен цвят да се окаже забранен достъп.

Таблица 3.5. Технически параметри на LED

Пад на напрежение – зелен цвят	1.9 V – 4 V
Пад на напрежение – червен цвят	1.6 V – 2 V
Работен ток	20 mA

### Интеграция на MFRC522 RFID модул

Използва се библиотека с отворен код [17], която предоставя лесен начин за работа с четеща. Комуникацията между микро-контролера и четеща се осъществява чрез Serial Peripheral Interface (SPI) [13] протокола.

Библиотеката има определени очаквания към разположението на щифтовете за да работи коректно.

Таблица 3.6. Разположение на щифтове при MRFC522

Сигнал	Четец	Ардуино
RST/Reset	RST	Digital 9
SPI SS	SDA	Digital 10
SPI MOSI	MOSI	Digital 11
SPI MISO	MISO	Digital 12
SPI SCK	SCK	Digital 13
GND	GND	GND
VCC	3.3 V	3.3 V

### Интеграция на ESP8266 Wi-Fi модул

Модула и притежава собствена изчислителна мощ (32-bit micro-CPU) и се разпространява с firmware зареден с AT Commands. [15]

Комуникацията между модула и ардуиното става чрез серийна комуникация на базата на universal asynchronous receiver-transmitter (UART). Тъй като ардуиното има само два серийни щифта: 0 – RX, 1 – TX, които се използват за връзка с компютър по – време на разработка и тестване на кода, се използва библиотека SoftwareSerial, която дава възможност за серийна комуникация чрез използване на другите цифрови щифтове. Библиотеката е част от ядрото на ардуиното, което представлява група библиотеки, които се разпространяват с него. [18]

Модула е свързан към ESP-01 адаптера, който от своя страна е свързан към ардуиното.

Таблица 3.7. Разположение на щифтовете при ESP8266

<b>ESP8266</b>	<b>ESP-01 адаптер</b>	<b>Ардуино</b>
RX	RX	Digital 6 (TX)
TX	TX	Digital 5 (RX)
GND	GND	GND
3.3 V	VCC	5 V
EN	VCC	5 V

## Принцип на работа

Bootloader-а на ардуиното ни позволява да подменяме изпълнявания се код без да се налага да използваме директно програматор. Има две основни функции, които ни предоставя bootloader-а за да работим с ардуиното, това са setup и loop функциите.

setup функцията се изпълнява само веднъж – при първоначалното захранване на ардуиното. Тук се поставя инициализиращ код.

loop функцията се изпълнява периодично, чрез нея имаме възможност да работим с текущото състояние на ардуиното през много кратък период, което позволява да реагираме на промени при различните щифтове и външните модули. Тук се поставя код, който следи за състоянието на съответните щифтове и външни устройства и взема съответните решения какво следва да се изпълни.

Скрипта зареден в ардуиното първоначално зарежда SoftwareSerial библиотеката за работа със серийна комуникация, след това се зарежда SPI библиотека за работа с SPI протокола и накрая се зарежда библиотека за работа с MFRC522 модула.

В setup функцията се инициализира променлива, която съдържа уникалния номер на ардуиното. В случая това е променлива, която се избира от

програмиста, тъй като ардуино платформата не дава възможност за достъп до серийна номер на устройството.

Цифрови щифтове 2 и 3 се инициализират като изходни, с цел захранване на зеления и червения LED.

SoftwareSerial библиотеката се настройва на цифрови щифтове 5 и 6 от ардуиното за серийна комуникация с Wi-Fi модула. Където 5-ти се използва за приемане, а 6-ти се ползва за изпращане на данни – огледално на тези на Wi-Fi модула. По – този начин изходните данни от модула ще се третират като входни за ардуиното, а изходните от ардуиното ще се третират като входни за модула.

Инициализира се URI адреса и порта на сървъра, които са част от скрипта, който се изпълнява. Тъй като те трябва да се знаят предварително, за текущата дипломна работа сървъра е конфигуриран на адрес 192.168.0.105 и порт 443. Изпраща се AT команда до Wi-Fi модула да създаде WLAN мрежа с име “arduino-master” и парола “test1234”, към която четеца се свързва с IP 192.168.0.100. Накрая на setup функцията червения и зеления LED се захранват в продължение на секунда за да се индикира, че инициализацията е преминала успешно.

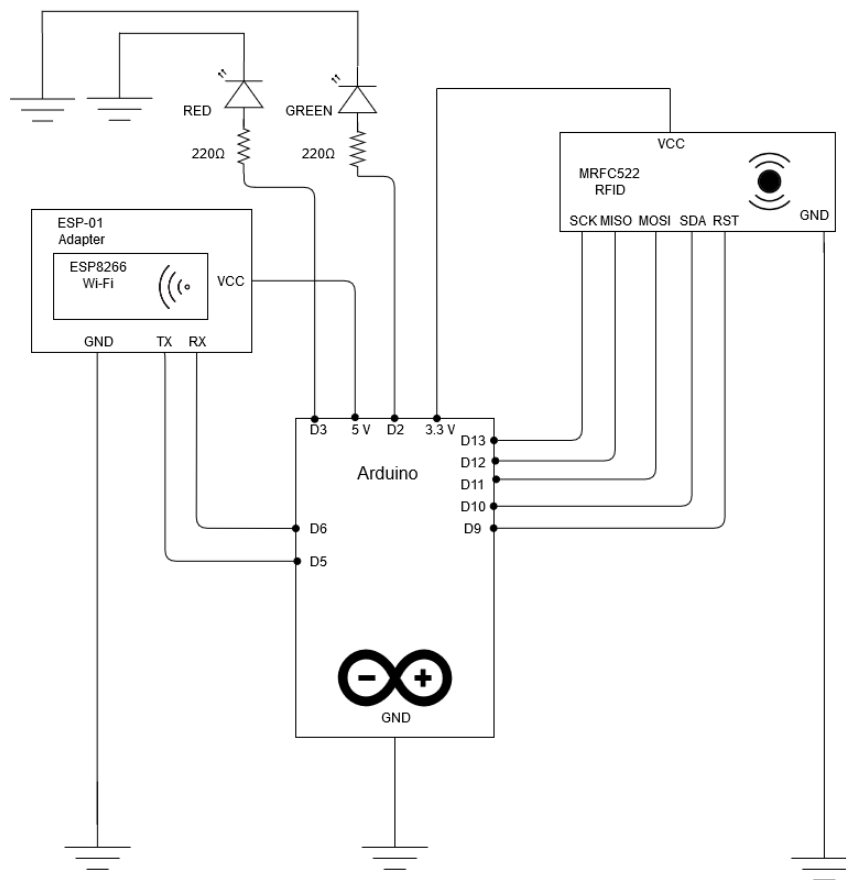
В loop функцията на всяка итерация се проверява дали четеца е успял да прочете данните на идентификатор в близост. При успешно засечен идентификатор, неговия уникален номер бива прочетен. Това става чрез използването на MFRC522 библиотеката. След това изпраща AT команда за отваряне на HTTPS връзка към сървъра и се изпраща заявка до “/accesscontrol/api/tags/checkaccess?TagNumber={tagNumber}&AccessPointSerialNumber={accessPointSerialNumber}”

където {tagNumber} е уникалният номер на идентификатора, а {accessPointSerialNumber} е уникалният номер на четеца.

Резултата от всяка AT команда връща съответно ОК за успешна операция и

ERROR при грешка, а при HTTP/HTTPS заявки се връща информация и за status code-а на response съобщението. Следва проверка за успешна операция и HTTPS заявка с status code 200, при което се захранва зеления LED за една секунда, а при не успешна проверка или не достатъчно ниво на достъп се захранва червения LED за една секунда.

### Схема на свързване

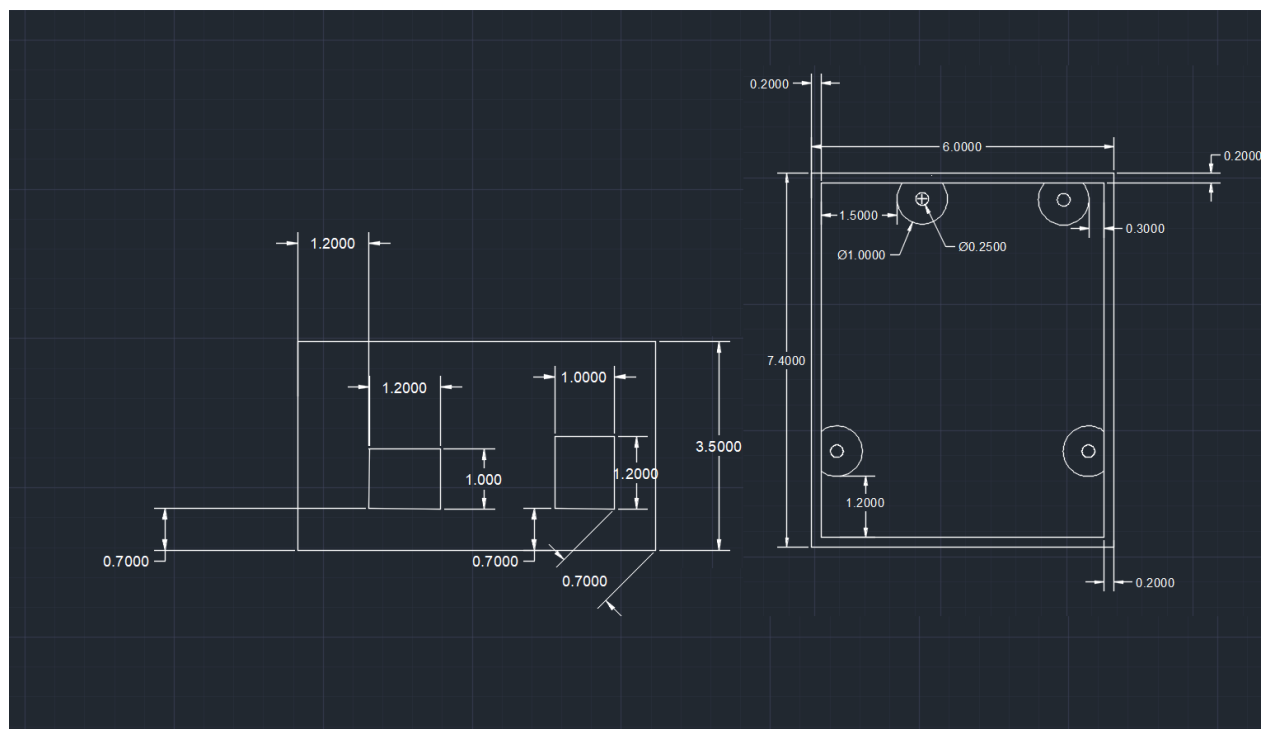


Фигура 3.6. Схема на свързване на четеца

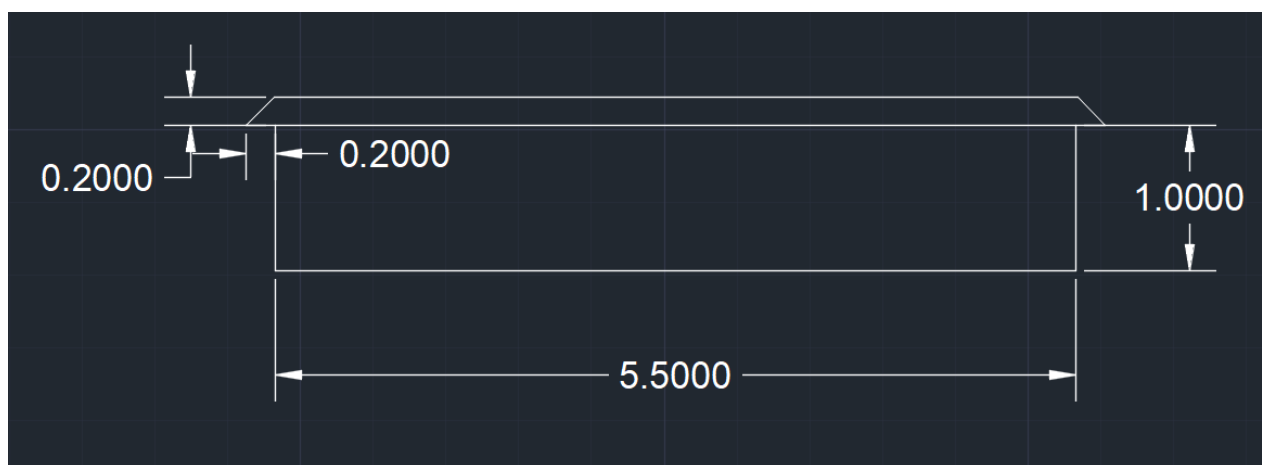
### Кутия

2D чертежите са направени на Autodesk AutoCAD LT 2020

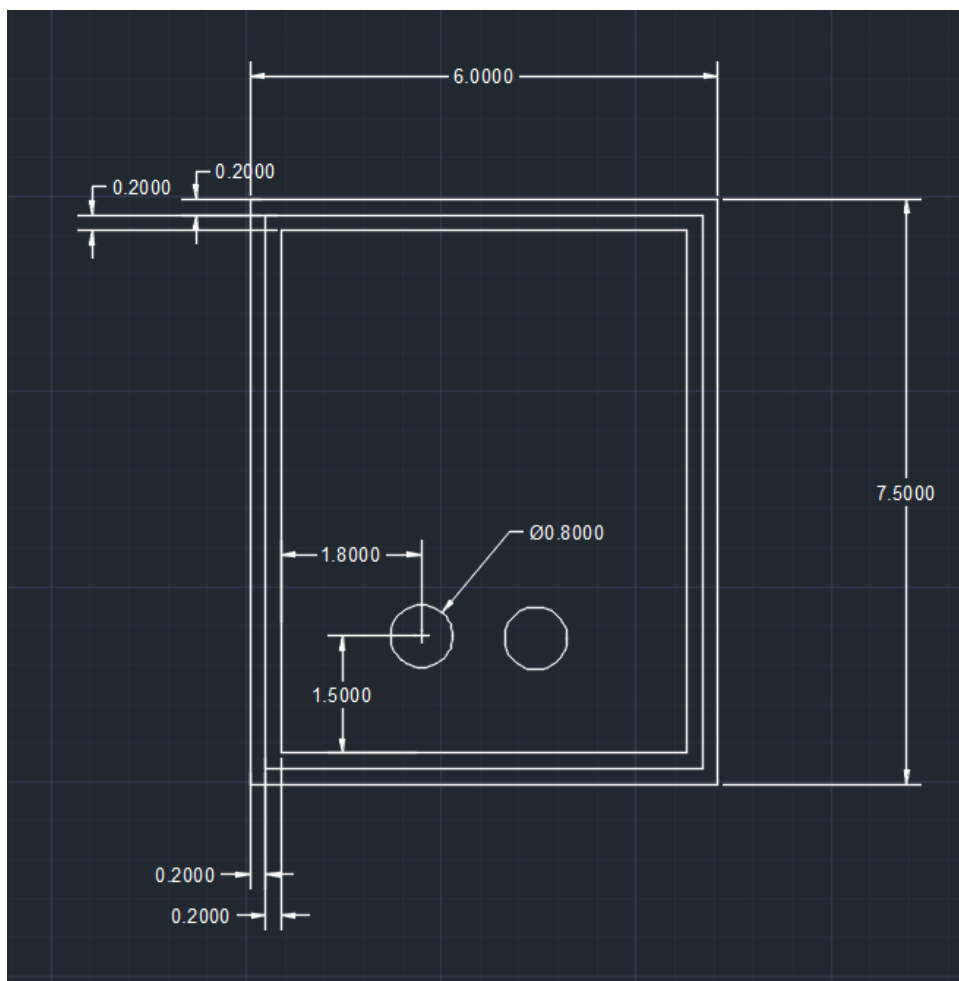




Фигура 3.7. 2D чертеж на кутията от пред (вляво) и отгоре (в дясно)

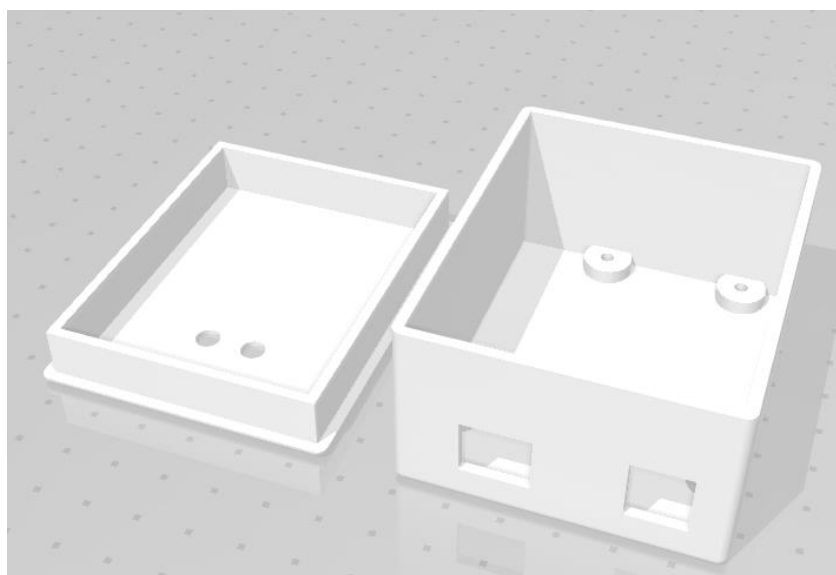


Фигура 3.8. 2D чертеж на капака на кутията гледан от пред



Фигура 3.9. 2D чертеж на капака на кутията гледан от горе

3D модела е направен на Autodesk 1234 Design



3.10. Фигура на 3D модел на кутията

## Финален продукт

Направена е с 3D принтер от 3D модела в предишната фигура



Фигура 3.11. Кутия принтирана от 3D принтер

### 3.2. REST API

Основната идея на API-то е да делегира работата с базата и да подготвя данните от базата в подходящ вид. Използва се както от Website-a за да се извършват всички административни дейности, както и от Ардуиното за да се извършва проверката за достъп.

Имплементирано е на C# 7.3 под .NET CORE 2.1. Използван е ASP NET Core framework-a, която е базирана на Model-View-Controller модела.

Архитектурно проекта е разбит на пет части: контрол на достъп (AccessControl), администрация (Administration), автентикация (Auth), логове (Log), статистики (Stat).

#### AccessControl

Отговаря за проверката за контрол на достъп. Тази част се състои само от един контролер със следните функционалности:

/accessControl/api/tags/checkAccess - Сравнява нивото на достъп на идентификатора дали е достатъчно за да бъде пропуснат през определена

точка на достъп (четец). Това е URI-то към което се обръща ардуиното за да получи информация за контрола на достъп. Всяко изпълнение на тази заявка в базата се генерира събитие, което запазва пълната информация за идентификатора и точката на достъп – впоследствие тези събития се използват за генерирането на справки.

## **Administration**

Отговаря за всички заявки свързани със сайта за администрация. Кода е разбит в четири контролера: `AccessPointController`, `ExportController`, `TagsController`, `UsersController`.

**AccessPointController** – дава възможност за работа с точките на достъп (четци)

[/administration/api/accessPoint/register](#) - Регистрира нова точка на достъп

[/administration/api/accessPoint/activate](#) - Активира неактивна точка на достъп

[/administration/api/accessPoint/deActivate](#) - Деактивира активна точка за достъп

[/administration/api/accessPoint/accessLevel](#) - Променя нивото на достъп за дадена точка на достъп|

[/administration/api/accessPoint/update](#) - Променя оказаните свойства на дадена точка на достъп. Ако някое от свойствата в заявката няма стойност се пренебрегва по време на update-a.

[/administration/api/accessPoint/delete](#) - Изтрива дадена точка за достъп

[/administration/api/accessPoint/undelete](#) - Възстановява изтрита точка на достъп

[/administration/api/accessPoint/active](#) - Връща страницирана информация за активните точки на достъп

[/administration/api/accessPoint/inactive](#) - Връща страницирана информация за всички неактивни точки на достъп

[/administration/api/accessPoint/deleted](#) - Връща страницирана информация за

всички изтрети точки на достъп

/administration/api/accessPoint/unknown - Връща страницирана информация за всички непознати точки на достъп

/administration/api/accessPoint/count - Връща информация за бройките на различните състояния на точките за достъп.

**ExportController** – дава възможност за експорт на данните.

/administration/api/export - Връща пълна информация за потребителите, точките на достъп, идентификаторите и събитията.

**UsersController** – дава възможност за работа администраторски потребители

/administration/api/users/register - Регистрира администраторски потребител.

**TagsController** – дава възможност за работа с идентификаторите.

/administration/api/tags/register - Регистрира нов идентификатор по номер, ниво на достъп и потребител на когото принадлежи.

/administration/api/tags/activate - Активира неактивен идентификатор.

/administration/api/tags/deactivate - Деактивира активен идентификатор.

/administration/api/tags/delete - Изтрива идентификатор.

/administration/api/tags/undelete - Възстановява изтрит идентификатор.

/administration/api/tags/accessLevel - Променя нивото на достъп на даден идентификатор

/administration/api/tags/update - Променя свойствата на идентификатора с оказаните в тялото на заявката. Ако някое от свойствата в заявката няма стойност се пренебрегва по време на update-a.

/administration/api/tags/active - Връща страницирана информация за активните идентификатори.

/administration/api/accessPoint/inactive - Връща страницирана информация за неактивните идентификатори.

/administration/api/accessPoint/unknown - Връща страницирана информация за непознатите идентификатори.

/administration/api/accessPoint/deleted - Връща страницирана информация за изтритите идентификатори

/administration/api/accessPoint/count - Връща информация за бройките на идентификаторите в различните състояния

/administration/api/accessPoint/users - Връща информация за всички потребители, които притежават идентификатори.

## **Auth**

Отговаря за всички функционалности свързани с автентикацията на потребителите.

/auth/api/token/generate - Генерира JWT по даден емайл адрес и парола на потребител.

## **Log**

Отговаря за всички функционалности свързани със записването на грешки възникнали при клиента.

/log/api/client/error - Записва информация за възникнали грешни при клиента.

## **Stat**

Отговаря за функционалности свързани с генерирането на справки

/stat/api/user/overview - Връща информация средностатистически когато потребителите идват на работа, тръгват от работа и средно с колко се отмества прекараното време на работното място спрямо очаквания осем часовия работен ден.

/stat/api/user/all - Връща информация за всеки един потребители - емайл адрес, средностатистически кога идва на работа, средностатистически кога си тръгва

от работа и с колко се отнема времето прекарано на работното място спрямо осем часовия работен ден.

/stat/api/user/norm - Връща информация за среднестатистическа информация за отнемането на изработените часове спрямо осем часове работен ден за даден потребител за определен период от време. Информацията е на дневна база.

/stat/api/user/entrance - Връща информация за среднестатистическа информация за това, когато за даден потребител идва на работа. Информацията е на дневна база.

/stat/api/user/exit - Връща информация за среднестатистическа информация за това, когато за даден потребител си тръгва от работа. Информацията е на дневна база.

### **3.3. Website**

Изграден е на базата на Angular framework-а. Основната градивна единица е така наречения компонент. Той представлява съвкупност TypeScript файл за дефиниране на логиката, HTML за дефиниране на визуализационната структура и CSS файл за дефиниране на дизайна на резултатния компонент.

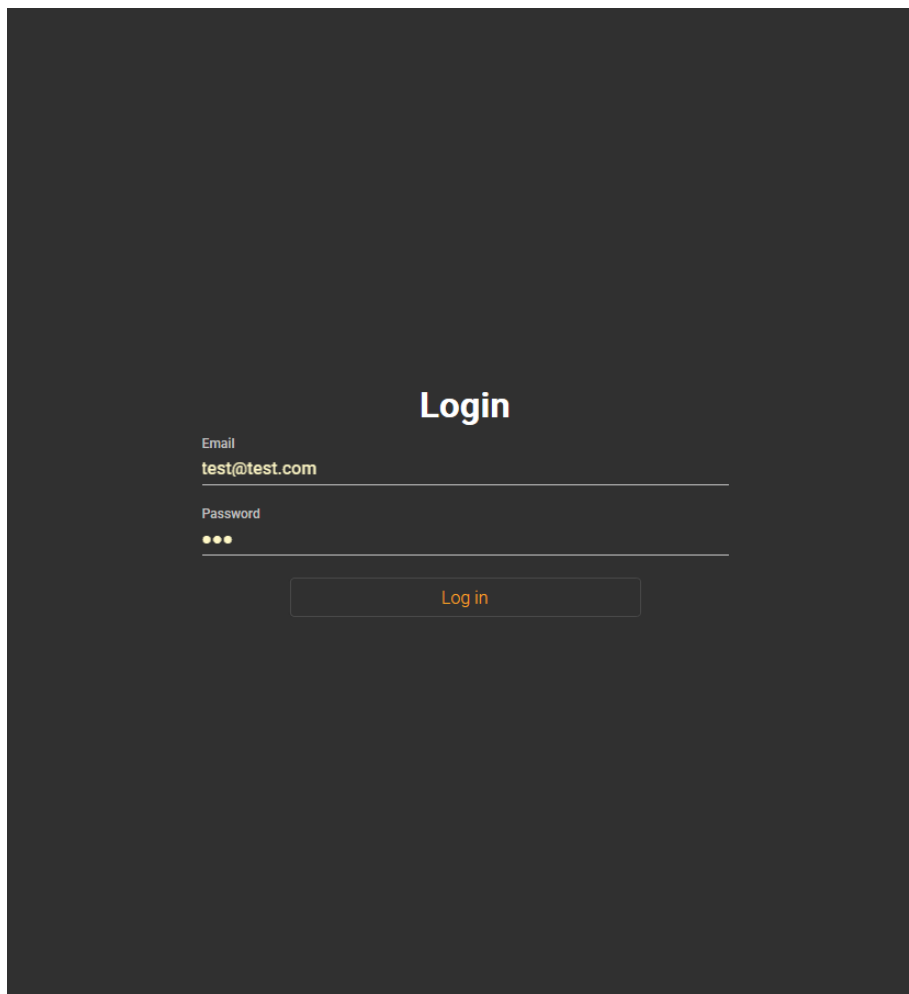
Компонентно базираните framework-ци позволяват по –добро разделение на кода, съответно по – добра изолация, което неминуемо води до по – лесната му поддръжка.

За дизайна е използвана Materialize библиотеката, която дефинира предварително готови компоненти, които се подчиняват на общи дизайна правила. Под компоненти се разбира: бутони, таблици списъци.

Website-а не е публично достъп, но за целта на дипломната работа website-а може да се отвори само от други компютри в същата мрежа в която се намира и сървър на адрес: <https://www.desktop-sb3j0h0/>.

За самата демонстрация е създаден и администраторски потребител със следните данни: **Емайл:** [test@test.com](mailto:test@test.com), **Парола:** 123.

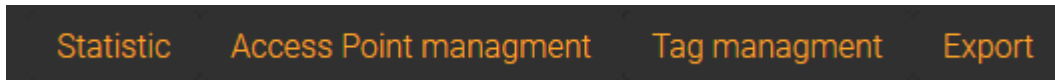
При първоначално отваряне на сайта в browser ще ви пренасочи към страница за автентикация, чрез въвеждане на потребителско име и парола.

A screenshot of a login page with a dark gray background. The word "Login" is centered at the top in white. Below it, there are two input fields. The first is labeled "Email" and contains the text "test@test.com". The second is labeled "Password" and contains three yellow dots. Below the password field is a "Log in" button with orange text.

След успешна автентикация, приложението ви пренасочва към началната страница. Генерирания JWT е с валидност от три минути, в рамките на които администратора няма ще бъде третиран като автентикиран при всяко негово действие. След изтичането на тези три минути, системата ще ви пренасочи отново към страницата за въвеждане на потребителско име и парола.



Всяка страница съдържа в горен десен ъгъл навигационно меню. Сайта е разбита на четири основни страници: Statistic, Access Point management, tag management, Export

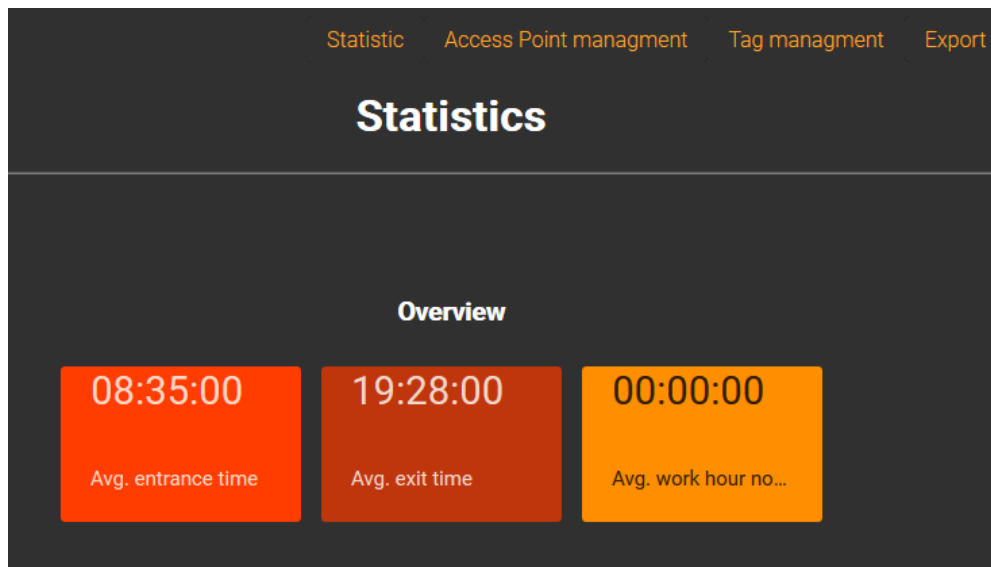


## Statistic

Показва справки свързани с потребителите свързани с това кога идват на работа, кога си тръгват от работа, по колко време прекарват на работа.

Съдържа три основни компонента: Overview, Users, Charts

Overview – визуализира информация за среднестатистически за всички потребители, кога потребителите идват на работа, кога тръгват и дали осем часовата норма за работния ден е изпълнена.



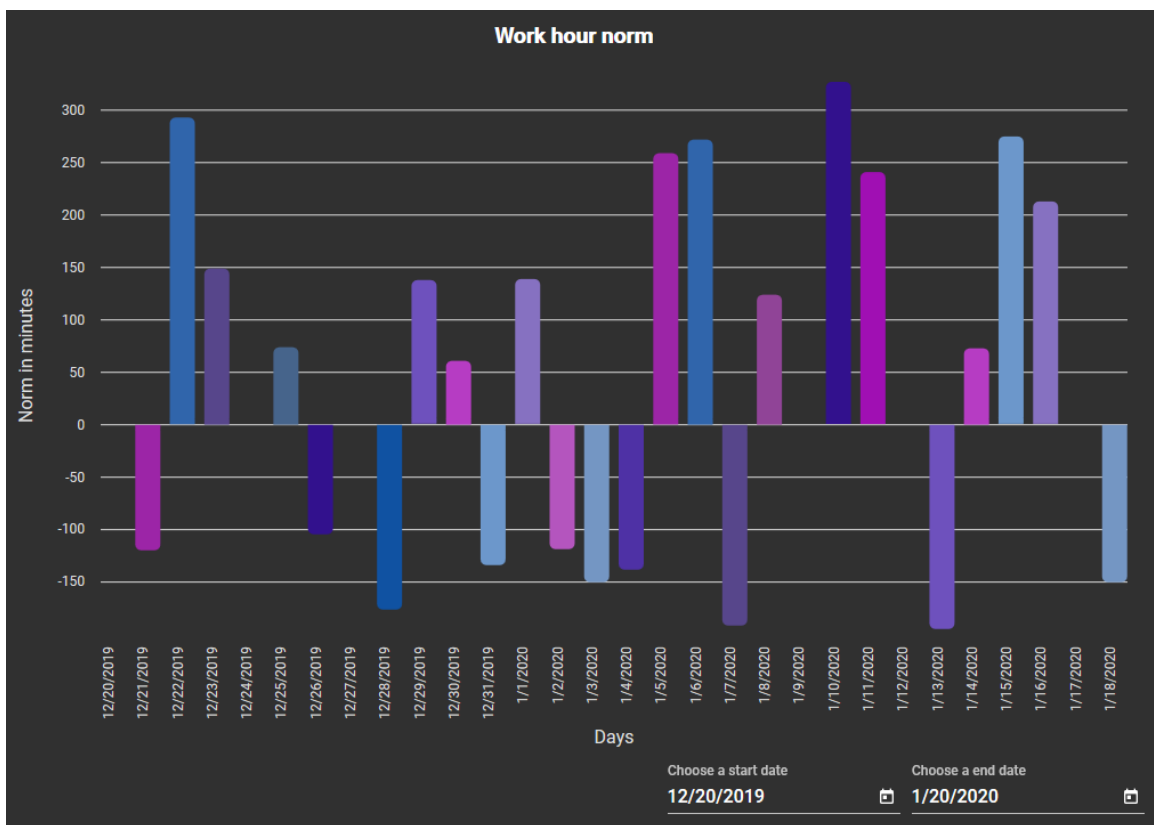
Users – визуализира за всички потребители среднестатистически кога идва на работа, кога си тръгва от работа и по колко време прекарва, под формата на списък.

Users			
Username	Avg. entrance time	Avg. exit time	Avg. Work norm
bratt bit	08:08:00	19:15:00	00:36:00
bratt round	08:25:00	19:38:00	00:39:00
goerge pike	08:33:00	20:02:00	01:13:00
goerge wood	08:00:00	19:07:00	00:37:00
harry wood	08:21:00	19:05:00	00:18:00
john bit	08:26:00	20:03:00	01:19:00
john davidson	09:01:00	18:53:00	-00:15:00
john malkovic	08:06:00	19:06:00	01:10:00
marry wood	08:42:00	19:53:00	00:55:00

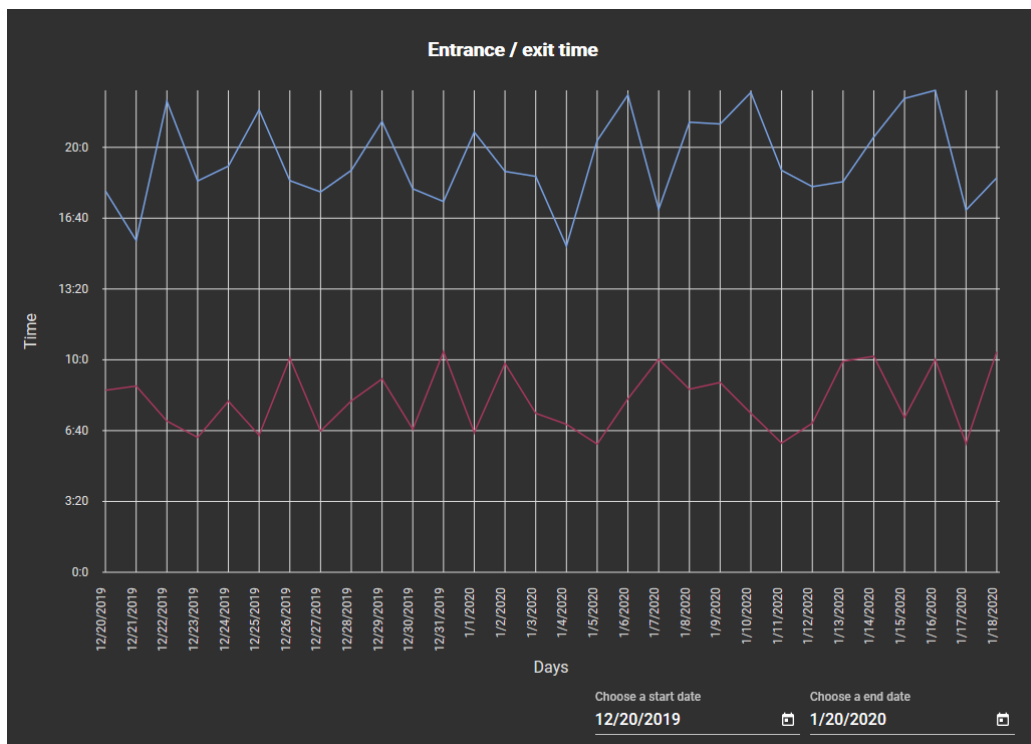
При селектиране на даден потребител от списъка, след него се визуализират две графики, свързани с данните на този потребител, като всяка от тях дава възможност ограничаване на времевия период за които да се генерират.

Първата графика показва времето прекарано на работа спрямо осем часов работен ден под формата на дневна база. Където положителна стойност означава, че потребителя е прекарал повече от осем часа на работа, а отрицателна означава, че не е успял да изкара осем часа на работа.

В сметките се взимат само действително прекараните часове на работното място, т.е. е предвидена възможността за обедна почивка или друг вид краткосрочно отсъствие от работното място.



Втората графика представлява съпоставяне между часа на идване на работа спрямо часа на тръгване от работа на дневна базата.



## Access point management

Визуализира списък с точките за достъп. Под точка на достъп се разбира четеца изграден на базата на ардуино. Точките на достъп могат да бъдат в четири различни състояния: Active (активно), In-Active (неактивно), Unknown (непознати за системата), Deleted (изтрети). Различните състояния са достъпни през ляво позиционирано странично меню.

Когато за първи се направи опит за достъп през дадена точка за достъп, тя бива регистрирана от системата като Unknown (непозната), отговорност е на администратора да активира и попълни нейните данни. Това е принципа за регистриране на точка за достъп в системата.

Всяка точка за достъп е представена от следните свойства:

**Access level** – минимално ниво на достъп нужно на даден потребител за да бъде пропуснат. Възможните стойности са Low, Mid, High

**Direction** – оказва дали точката на достъп е входна или изходна, това е от значение при генерирането на справките от Statistic частта.

**Description** – кратко описание за точката на достъп. Тук може да се впише на в коя сграда се намира и на кой етаж.

**Create Date** – да на създаване на точката за достъп в системата

**Modification Date** – дата на последна промяна в системата.

Следните операции могат да бъдат изпълнени върху дадена точка за достъп:

**Activate / De-Activate** – активиране или деактивиране на точка за достъп. Не активните точки за достъп не позволяват достъп на нито един потребител, независимо от техните нива на достъп.

**Delete** – изтрива точка на достъп. Изтритата точка на достъп също не позволява достъп на нито един потребител. Разделението между изтрита и неактивна точка на достъп е чисто административна, а не толкова

функционална. Когато дадена точка за достъп се повреди или вече не се ползва би било подходящо тя да бъде изтрита, докато деактивирането е по – подходящо за временно спиране на достъпа.

**Edit** – позволява редакция на горе изброените свойства.

The screenshot displays the 'Access point management' interface. On the left, a sidebar lists the status of access points: 'Active (9)', 'In-Active (12)', 'Unknown (0)', and 'Deleted (0)'. The main area is titled 'Active' and shows a configuration form for a specific access point with ID 'd38fabb5-b82a-4426-ba06-878272a119db'. The form includes fields for 'Access level' (set to 'Low') and 'Description' ('Building 6, entrance C floor 6, north side'). There is also a 'Direction' dropdown set to 'Entrance' with a note 'Min. 1 character length required'. At the bottom, it shows the 'Create date' (1/18/2020, 11:52:07 AM) and 'Modification date' (1/19/2020, 11:20:52 PM). Action buttons at the bottom right are 'De-Activate', 'Delete', 'Edit', and 'Save'.

## Tag management

Визуализира списък с идентификаторите. Идентификаторите идентично на точките за достъп също имат четири възможни състояния: Active (активни), In-Active (неактивни), Unknown (непознати), Deleted (изтрита). Различните състояния са достъпни през ляво позиционирано странично меню.

Аналогично на точката за достъп, регистрацията на нови идентификатори се случва при първи опит за достъп през определена точка с даден

идентификатор. Отговорността е на администратора да попълни данните за идентификатора и да го активира.

Всеки идентификатор притежава следните свойства:

**User** – името на собственика на идентификатора

**Access Level** – ниво на достъп на идентификатора

**Create Date** - дата на създаване на идентификатора в системата

**Modification Date** – последна промяна на идентификатора в системата

Следните операции могат да бъдат изпълнени за даден идентификатор:

**Activate / De-Activate** – активиране или деактивиране на идентификатор.

Деактивирания идентификатор няма право на достъп през която и да е точна на достъп.

**Delete** – изтриване на идентификатор

**Edit** – редактиране на горе посочените свойства на идентификатора

The screenshot displays a web interface titled "Tag management". On the left, there is a sidebar with four categories: "Active (18)" (highlighted in orange), "In-Active (0)", "Unknown (0)", and "Deleted (0)". The main content area is titled "Active" and shows a list of items. The first item is selected, displaying its ID: "43360620-050e-439d-9921-d83c0802cbf1". Below the ID, there are two input fields: "User\*" with the value "scarlet malkovic" and "Access level:" with a dropdown menu set to "Low". A message "Username required" is visible below the user field. At the bottom, there are four buttons: "De-Activate", "Delete", "Edit", and "Save". The "Create date" is "1/18/2020, 11:52:19 AM" and the "Modification date" is "--/--".

### 3.4. База данни

Използва се SQL Server Express Edition. Базата е разделена на четири логически груби функционалности под формата на схеми: access\_control, administration, log, stat, като във всяка схема се съдържат съответните таблици, функции, процедури и тригери.

**[access\_control].[AccessLevels]** – таблица съдържаща възможните нива на достъп за идентификаторите и точките на достъп. Биват три вида: Low (ниско), Mid (средно), High (високо)

Id – уникален пореден номер на нивото на достъп

Name – името на нивото на достъп (Low, Mid, High)

**[access\_control].[Direction]** – таблица съдържаща възможните ориентации на точките на достъп: Entrance (вход), Exit (изход)

Id – уникален номер на ориентацията на точките на достъп

Name – име на ориентацията (Entrance, Exit)

**[access\_control].[AccessPoints]** – таблица съдържаща информация за точките на достъп.

Id – уникален номер на точката на достъп

Description – описание за точката на достъп

SerialNumber – уникален номер с който се идентифицира точката на достъп

IsActive – показва дали точката на достъп е активна

IsDeleted – показва дали точката на достъп е изтрита

CreateDate – дата на създаване на записа

ModificationDate – дата на последна промяна

LevelId – навигационен ключ към таблицата за ниво на достъп.

DirectionId – навигационен ключ към таблицата за ориентация

**[access\_control].[Users]** – таблица съдържаща информация

Id – уникален номер на потребителя

Name – име на потребителя

**[access\_control].[Tags]** – таблица съдържаща информация за идентификаторите

Id – уникален номер на идентификатора

Number – уникален номер с който се идентифицира идентификатора

LevelId – навигационен ключ таблицата с нива на достъп

IsActive – показва дали идентификатора е активен

IsDeleted – показва дали идентификатора е изтрит

CreateDate – дата на създаване на записа в базата

ModificationDate – дата на последна промяна

UserId – навигационен ключ към таблицата с потребители, регистрирани в системата като притежатели на идентификатори

**[access\_control].[UnKnownAccessPoints]** – съдържа информация за непознатите точки на достъп. Това е всяка точка на достъп, която е направила заявка към сървъра, но няма информация за нея.

Id – уникален номер на непознатата точка за достъп

SerialNumber – уникален номер по който се идентифицира

AccessDate – последна дата към която е правен опит за достъп през непознатата точка за достъп

IsDeleted – показва дали е изтрита

**[access\_control].[UnknownTags]** – съдържа информация за непознатите идентификатори. Това е всеки идентификатор, който е бил използван за опит да се получи достъп без да има информация за него в базата данни.



**Id** – уникален номер на непознатия идентификатор

**AccessDate** – последна дата към която е оправен опит за достъп с идентификатора

**IsDeleted** – показва дали е изтрит

**[administration].[Roles]** – съдържа информация за ролите на администраторските потребители. Ролята на този етап е само една: Admin (администратор)

**Id** – уникален номер на ролята

**Name** – име на ролята (Admin)

**[administration].[Users]** – съдържа информация за администраторските потребители.

**Id** – уникален номер на потребителя

**Email** – емайл на потребителя

**PasswordHash** – hash-рана версия на паролата на потребителя

**CreateDate** – дата на създаване на записа

**ModificationDate** – дата на последна промяна на записа

**[administration].[UsersRoles]** – междинна таблица описваща връзка много към много между потребителите и ролите.

**UserId** – навигационен ключ към таблицата с потребители

**RoleId** – навигационен ключ към таблицата с роли

**[log].[Type]** – съдържа информация за видовете логове, които се записват в базата от страна на клиента. Следните типове са налични: Error (грешка)

**Id** – уникален номер типа

**Name** – име на типа (Error)

[stat].[Events] – съдържа информация за идентификаторите и точките на достъп за конкретен момент за извършен опит на даден потребител да премине през определена точка на достъп. Чрез данните в тази таблица се генерират справките за даден потребител.

Id – уникален номер на събитието

TagNumber – идентификационен номер на идентификатора

TagLevelId – ниво на достъп на идентификатора

TagIsActive – показва дали идентификатора е бил активен

TagIsDeleted – показва дали идентификатора е бил изтрит

TagIsUnknown – показва дали идентификатора дали е бил непознат

UserId – навигационен ключ към таблицата с потребители, които са притежатели на идентификатор

AccessPointSerialNumber – идентификационен номер на точката за достъп

AccessPointLevelId – навигационен ключ към таблицата с нива за достъп

AccessPointDirectionId – навигационен ключ към таблицата за ориентация

AccessPointIsActive – показва дали точката за достъп е била активна

AccessPointIsDeleted – показва дали точката за достъп е била изтрит

AccessPointIsUnknown – показва дали точката за достъп е била позната

CreateDate – дата на създаване на събитието

## Глава 4. Анализ на получените резултати, приложимост и изводи

При анализ на една цялостна система за контрол на достъпа може да се разгледат множество характеристики, като по – важните са:

1. Бързодействието на системата
2. Сигурността
3. Капацитета на системата
4. Гъвкавост при интегриране

Това ще даде ясна представа къде са силните и слабите страни на системата и възможните посоки за подобрене на им.

Използван hardware на сървърната част: Intel Core i5-3320M 2.60 GHz (2 ядра), 8 GB RAM, SSD 120 GB, Windows 10

### 4.1. Бързодействие на системата

#### Сървър

Направен е тест на сървърната част представляващ две паралелни заявка през няколко секунди в продължение на една минута. Теста има за цел да симулира ситуация в която имаме сграда с два входа през които потребителите се автентикират. Резултат трябва да покаже, че REST API-то не изисква сериозни ресурси и работи с лекота при нормални условия.

Теста е реализиран чрез разработка на конзолно приложение, което изпраща заявка до сървъра.

**Резултата:** средното време за което се обработва 1 заявка докато системата с натоварване от 2 паралелни заявки е 53 ms.

Следва натоварващ тест имащ за цел да покаже, възможността на системата да се справи с по – сериозен обем от потребители. Теста изпраща 200 паралелни заявки до сървъра в продължение на една минута. Теста симулира сценарии в които имаме 200 точки за контрол на достъп, като през всяка една през 1 секунда има потребител.

**Резултат:** средното време обработка на една заявка докато системата е натоварена с 200 паралелни заявки е 88 ms.

Извода, който може да се направи е, че REST API-то изисква минимален ресурс и е подходяща дори и за крайно натоварени обекти.

## Четец

Направен е тест на бързодействието на четеца. В това се включва времето нужно за прочитане на идентификатора, изпращането на данните до сървър, връщане на отговор от сървър – тества се цялостното бързодействие на системата.

За да се отчете времето се ползва `millis()` функцията, която е част от библиотеката на ардуино. Тя връща текущото време от стартирането на ардуиното в милисекунди. Измерва се времето преди прочитане на идентификатора и времето след индикацията за отговор от сървър и накрая се прави разлика между двете за да се изчисли изминалото време.

**Резултат:** Направени са 10 заявки до сървър и осреднената стойност е 1500 ms.

Според изследване направено за определяне на човешкото възприятие от гледна точка на работа с компютър стигат до извода, че забавяне от 100 ms е неуловимо за потребителя, а закъснение от 1000 ms е горната граница в която потребителя не би се разсеял, независимо че ще забележи забавянето. [19]

Времето към което трябва да се стремим е около 1000 ms, от което може да направим извода, че скоростта на четеца е приемлива, но не задоволителна - допълнително усилие в тази посока е желателно. Възможно решение би било усилване на сигнала от антената на четеца с което да се увеличи максималното разстояние на комуникация между четеца и идентификатора.

## 4.2. Сигурност

### Връзка между четец и сървър

Цялата комуникация между четеца и сървър се осъществява на базата на HTTPS. След анализ на трафика между четеца и сървър посредством WireShark, се вижда, че комуникацията е HTTPS и е избрана версия на протокола TLS 1.1.

Time	Source	Destination	Protocol	Length	Info
1 0.000000	192.168.0.100	192.168.0.105	TCP	58	5454 → 443 [SYN] Seq=0 Win=2920 Len=0 MSS=1460
2 0.000138	192.168.0.105	192.168.0.100	TCP	58	443 → 5454 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3 0.006625	192.168.0.100	192.168.0.105	TLSv1.1	110	Client hello
4 0.007214	192.168.0.105	192.168.0.100	TLSv1.1	917	Server Hello, Certificate, Server Hello Done
5 0.102924	192.168.0.100	192.168.0.105	TLSv1.1	396	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
6 0.104859	192.168.0.105	192.168.0.100	TLSv1.1	129	Change Cipher Spec, Encrypted Handshake Message
7 0.233994	192.168.0.100	192.168.0.105	TCP	54	5454 → 443 [ACK] Seq=399 Ack=939 Win=1982 Len=0
8 0.355511	192.168.0.100	192.168.0.105	TLSv1.1	251	Application Data
9 0.396885	192.168.0.105	192.168.0.100	TCP	54	443 → 5454 [ACK] Seq=939 Ack=596 Win=63645 Len=0
10 0.502082	192.168.0.105	192.168.0.100	TLSv1.1	203	Application Data
11 0.572528	192.168.0.100	192.168.0.105	TCP	54	5454 → 443 [FIN, ACK] Seq=596 Ack=1088 Win=1833 Len=0
12 0.572682	192.168.0.105	192.168.0.100	TCP	54	443 → 5454 [FIN, ACK] Seq=1088 Ack=597 Win=63645 Len=0
13 0.576231	192.168.0.100	192.168.0.105	TCP	54	5454 → 443 [ACK] Seq=597 Ack=1089 Win=1832 Len=0

Фигура 4.1. Снимка на пакетите между четеца и сървър анализирани чрез WireShark

TLS 1.1 стандарта е създаден през 2006 година. Към текуща дата протокола се води остарял, също така водещите browser-и съветват преминаването към последна версия 1.2 на протокола. [20]

Последната версия на firmware-а на ESP8266 Wi-Fi модула има поддръжка за TLS 1.2.

За да се подsigури сигурна връзка между четеца и сървъръ е необходим update до последна версия на firmware-а на ESP8266 Wi-Fi.

## Връзка между website и сървър

Тук комуникацията също е изцяло на HTTPS, като протокола, който се използва ще зависи пряко от версията на операционната система и browser.

С Windows 10 и FireFox v72.0.1 се използва TLS 1.2.

43	25.110351	fe80::c4c2:11df:d74b:5bd2	fe80::81ff:1d66:86bf:4049	TCP	74 61706 → 443 [ACK] Seq=1 Ack=1 Win=66048 Len=0
44	25.113367	fe80::c4c2:11df:d74b:5bd2	fe80::81ff:1d66:86bf:4049	TLSv1.2	591 Client Hello
45	25.126998	fe80::81ff:1d66:86bf:4049	fe80::c4c2:11df:d74b:5bd2	TLSv1.2	1322 Server Hello, Certificate, Server Key Exchange, Server Hello Done
46	25.235699	fe80::c4c2:11df:d74b:5bd2	fe80::81ff:1d66:86bf:4049	TLSv1.2	232 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
47	25.237465	fe80::81ff:1d66:86bf:4049	fe80::c4c2:11df:d74b:5bd2	TLSv1.2	125 Change Cipher Spec, Encrypted Handshake Message
48	25.237579	fe80::81ff:1d66:86bf:4049	fe80::c4c2:11df:d74b:5bd2	TLSv1.2	143 Application Data
49	25.238732	fe80::c4c2:11df:d74b:5bd2	fe80::81ff:1d66:86bf:4049	TLSv1.2	251 Application Data
50	25.241002	fe80::c4c2:11df:d74b:5bd2	fe80::81ff:1d66:86bf:4049	TLSv1.2	398 Application Data
51	25.241052	fe80::81ff:1d66:86bf:4049	fe80::c4c2:11df:d74b:5bd2	TLSv1.2	112 Application Data
52	25.251498	fe80::c4c2:11df:d74b:5bd2	fe80::81ff:1d66:86bf:4049	TLSv1.2	161 Application Data
53	25.263622	fe80::c4c2:11df:d74b:5bd2	fe80::81ff:1d66:86bf:4049	TCP	74 61706 → 443 [ACK] Seq=1264 Ack=1369 Win=64768 Len=0
54	25.265590	fe80::c4c2:11df:d74b:5bd2	fe80::81ff:1d66:86bf:4049	TLSv1.2	112 Application Data
55	25.265641	fe80::81ff:1d66:86bf:4049	fe80::c4c2:11df:d74b:5bd2	TCP	74 443 → 61706 [ACK] Seq=1407 Ack=1302 Win=131072 Len=0
56	25.376468	fe80::c4c2:11df:d74b:5bd2	fe80::81ff:1d66:86bf:4049	TCP	74 61706 → 443 [ACK] Seq=1302 Ack=1407 Win=64768 Len=0
57	26.346802	fe80::81ff:1d66:86bf:4049	fe80::c4c2:11df:d74b:5bd2	TLSv1.2	201 Application Data

Фигура 4.2. Снимка на пакетите между browser и сървъръ анализирани чрез WireShark

## 4.3. Капацитет на системата

### Брой връзки

От гледна точка на максимален брой паралелни заявки това зависи пряко от инфраструктурата: сървър и рутер. REST API-то само по себе си няма зададени ограничения.

### Размер на базата данни

Направен е анализ на максималния размер на един запис в базата за таблиците, които се очаква да нарастват във времето. Това ще ни даде ясна представа за нуждите на системата от към дисково пространство.

За да се направи анализа се използва функция от SQL Server, която предоставя информация за размерите на редовете в дадена таблица

`dbcc showcontig ('{table_name}')` with `tableresults`

Трите таблици които се очаква да нарастват са:

[access\_control].[Tags] – пази информация за идентификаторите. Максимален размер за един запис е 112 bytes.

[access\_control].[AccessPoints] – пази информация за точките на достъп. Максимален размер за един запис е 213 bytes.

[stat].[Events] – пази информация за всеки опит за достъп осъществен през дадена точка на достъп и идентификатор. Максимален размер за един запис е 187 bytes.

За да имаме реална постановка при изчислението на размерите ще разгледаме обект с 500 точки за достъп и 50 000 идентификатора. Подобна система би могла да бъде фирма за логистика.

$500 \times 112 \text{ bytes} = 54 \text{ KB}$

$50\,000 \times 213 \text{ bytes} = 10.65 \text{ MB}$

За да се пресметне размера на последната таблица ще вземем по – реален пример. Размера зависи изцяло от броя опити за проверка на контрол на достъп.

За пример ще вземем данните за натоварването на метро-станциите в Торино, Италия. Според публична информация за един ден преминат 155 000 пътника, а за година 41 милиона. [21]

$41\,000\,000 \times 187 \text{ bytes} = 7.67 \text{ GB}$  на годишна база

В днешно време твърдите дискове варират в размери от 20 GB до 12 TB, като при средностатистическия потребител най – често срещаните варират от 200 GB до 2 TB, а сървърните са най – често над 1 TB. Извода, който може да се направи, е че системата няма сериозни изисквания от към дисково пространство и продължителното използване на системата дори и в крайно натоварени обекти би била покрита от днешните стандарти за твърди дискове.

#### **4.4. Гъвкавост при интегриране**

REST API-то и website-а са написани на езици за програмиране, които са cross-platform. Това позволява интегрирането им на всички водещи операционни системи като Windows, Linux, Mac OS, Darwin.

## Заклучение

Настоящата дипломна работа разглежда принципите на работа в една система за контрол на достъп с административен модул базирана на RFID технологията посредством Arduino микро-контролер за четец и безжична комуникация към сървърна част, както и цялостно изграждане, и изпробване на системата в реални условия.

Системата е направена с цел висока сигурност, бързодействие и възможност за анализ на данните.

Безжичната комуникация, голямата модулност на Arduino-то, както и използването на широко разпространени езици за програмиране, голямата екосистема под формата на библиотеки и модули, предоставят сериозни удобства при изграждането и поддръжката на четеца.

Минималните изисквания, интеграция върху широк набор от операционни системи и възможността за използване на вече изградени инфраструктурни решения като сървъри и безжични мрежи, улеснява значително интегрирането на REST API-то и административния website.

Съвременни технологии като HTTP, безжични мрежи и RFID ще продължат да бъдат активно развивани и подобрявани, което ги прави правилния избор за изграждането на една такава система.

## Литература и интернет източници

C# documentation <https://docs.microsoft.com/en-us/dotnet/csharp/>

Arduino programming language reference <https://www.arduino.cc/reference/en/>

[1]. Stanimir S. Valtchev, Elena N. Baikova, Luis R. Jorge (2012, December 10). Electromagnetic fields as the wireless transporter of energy. Elec. Energy. Vol 25, N 3, pp. 171 – 181

[2] JSON в Introducing JSON  
<https://www.json.org/json-en.html>

[3] SQL Server protocols в Microsoft Docs  
[https://docs.microsoft.com/en-us/openspecs/sql\\_server\\_protocols/ms-sqlprotlp/f16558b2-4561-45be-89c9-6f9114514c97](https://docs.microsoft.com/en-us/openspecs/sql_server_protocols/ms-sqlprotlp/f16558b2-4561-45be-89c9-6f9114514c97)

[4] IEEE 802.11 стандарт в IEEE Standard for information technology  
[https://standards.ieee.org/standard/802\\_11-2016.html](https://standards.ieee.org/standard/802_11-2016.html)

[5] Wi-Fi в Wikipedia, The Free Encyclopedia  
<https://en.wikipedia.org/wiki/Wi-Fi>

[6] TCP/IP в Wikipedia, The Free Encyclopedia  
[https://en.wikipedia.org/wiki/Internet\\_protocol\\_suite](https://en.wikipedia.org/wiki/Internet_protocol_suite)

[7] HTTP в MDN  
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>

[8] HTTP полета (headers) в Wikipedia, The Free Encyclopedia  
[https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_header\\_fields#Request\\_fields](https://en.wikipedia.org/wiki/List_of_HTTP_header_fields#Request_fields)

[9] JWT в JSON Web Token Introduction  
<https://jwt.io/introduction/>



[10] HMAC в Wikipedia, The Free Encyclopedia

<https://en.wikipedia.org/wiki/HMAC>

[11] Base64 в Wikipedia, The Free Encyclopedia

<https://en.wikipedia.org/wiki/Base64>

[12]. Arduino Uno REV3 в Arduino Official Store

<https://store.arduino.cc/arduino-uno-rev3>

[13]. Contactless RFID reader / writer MRFC522 (2016, Април 27) в NXP semiconductors

<https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf>

[14]. MIFARE ISO14443A key tag в NXP

[https://www.nxp.com/docs/en/data-sheet/MF1S50YYX\\_V1.pdf](https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf)

[15]. Wi-Fi module ESP8266 в NURDspace

<https://nurdspace.nl/ESP8266>

[16] ESP-01 адаптер и регулятор на напрежение в ProtoSuppliers

<https://protosupplies.com/product/esp8266-esp-01-adapter-with-voltage-regulator/>

[17] MRFC522 RFID библиотека с отворен код в GitHub

<https://github.com/miguelbalboa/rfid>

[18] SoftwareSerial библиотека в Arduino

<https://www.arduino.cc/en/Reference/softwareSerial>

[19] Miller, R. B. (1968). Response time in man-computer conversational transactions. Proc. AFIPS Fall Joint Computer Conference Vol. 33, 267-277.

[20] TLS 1.0 и TLS 1.1 ще бъдат третираны като несигурни от 2020 в BleepingComputer

<https://www.bleepingcomputer.com/news/security/tls-10-and-tls-11-being-retired-in-2020-by-all-major-browsers/>

[21] Интензивност на метро-станциите в Турино в Turin metro map, Italy  
<http://mapa-metro.com/en/italy/turin/turin-metro-map.htm>

## Използвани съкращения

**СКД** – система за контрол на достъп.

**Wi-Fi (Wireless Fidelity)** – безжична локална мрежа, която използва радио вълни, без нуждата от кабели.

**RFID (Radio-frequency identification)** – Радиочестотна идентификация.

**IDE (от Integrated Development Environment)** – среда за разработка на софтуер.

**API (Application Programming Interface)** – интерфейс или протокол за комуникация между отделни компютърни системи с цел по – лесно разработка, поддръжка и преизползваемост.

**REST (Representational state transfer)** – стандарт за достъп и манипулация на ресурси лежащ на HTTP(S) протокола.

**HTTP (Hypertext transfer protocol)** – протокол за обмяна на данни заложен в основите на световната мрежа (World Wide Web)

**URI (Uniform Resource Identifier)** – текстов формат за идентифициране на даден ресурс

**HTML (Hypertext markup language)** – Xml базиран формат използван за изграждането на web страници.

**XML (Extensible markup language)** - вид текстов формат за описване на данни.

**HTTPS (Hypertext transfer protocol secure)** – разновидност на HTTP протокола с цел по – висока сигурност

**LAN (Local area network)** – компютърна мрежа за свързване на устройства в дадена среда.

**ТСР/ІР** – група от комуникационни протокол за обмяна на информация под формата на пакети и ІР адреси.

**TCP** (Transmission Control Protocol) – един от основните протоколи в TCP/IP модела

**IP** (Internet Protocol) – един от основните протоколи в TCP/IP модела

**WWW** (Word wide web) – световната интернет мрежа

**JWT** (JSON Web Toke) – стандарт използван за автентикация

**Transport Later Security** (TLS) – протокол за криптиране на трафика част от TCP/IP стандарта

## **Изходен код и използван software**

Кода на приложението е публично достъпен на:

<https://github.com/vasiloreshenski/RFID>

Използван software:

**Visual Studio 2017 Community Edition** – за разработка на REST API-то

**Visual Studio Code** – за разработка на website-a

**Arduino IDE** – за разработка на кода изпълняван от ардуиното

**SQL Server Expression Edition** – база данни

**Internet Information Service (IIS)** – web сървър за обслужване на website-a и REST API-то.

**Fiddler** – за тестване на заявките по време на разработка на REST API-то.

**Notepad++** - текстов редактор

**AutoDesk 123D Design** – за 3D моделиране на кутията

**Autodesk AutoCAD LT 2020** – за проектиране на 2D схема на кутията

**WireShark** – за проследяване на TCP/IP трафика към сървъра